DevOps Shack

Kubernetes Tips & Useful Tricks With Usecases | Part-1,2,3,4,5,6,7

Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps

Part 1: Basic Kubernetes Commands & Tricks

1. Viewing Cluster Information

```
kubectl cluster-info
```

- **Use Case:** Quickly get addresses of the master and services.

2. Get Resources

```
kubectl get all
```

- **Use Case:** List all resources in the current namespace.

3. Using Kubectl Autocomplete

- **Setup for Bash:**

```
source <(kubectl completion bash)
```

- **Setup for Zsh:**

```
source <(kubectl completion zsh)
```

- **Use Case:** Speed up command entry with autocomplete.

4. Delete All Resources in a Namespace

```
kubectl delete all --all
```

- **Use Case:** Clean up a namespace for a fresh start.

5. Stream Pod Logs

```
kubectl logs -f [POD_NAME]
```

- **Use Case:** Follow log output in real time.

6. Execute Commands Inside a Pod

```
kubectl exec -it [POD_NAME] -- /bin/bash
```

- **Use Case:** Access the shell inside a pod.

7. Quickly Create and Expose a Pod

```
kubectl run mynginx --image=nginx --restart=Never --port=80
kubectl expose pod mynginx --port=80 --type=NodePort
```

- **Use Case:** Rapidly deploy and expose a simple application.

8. List All Pods in All Namespaces

```
kubectl get pods --all-namespaces
```

- **Use Case:** Overview of all pods across the cluster.

9. Output in YAML Format

```
kubectl get pod [POD_NAME] -o yaml
```

- **Use Case:** Get detailed configurations of resources.

10. Scale a Deployment Quickly

```
kubectl scale deployment [DEPLOYMENT_NAME] --replicas=10
```

- **Use Case:** Modify the number of replicas dynamically.

11. Rollout History of a Deployment

```
kubectl rollout history deployment/[DEPLOYMENT_NAME]
```

- **Use Case:** Check the history and revisions of a deployment.

12. Undo a Deployment to a Previous State

```
kubectl rollout undo deployment/[DEPLOYMENT_NAME]
```

- **Use Case:** Revert to a previous deployment state if issues occur.

13. Apply Configuration From a File

```
kubectl apply -f [CONFIG_FILE.yaml]
```

- **Use Case:** Deploy or update resources in bulk.

14. Get Resource Manifest by Labels

```
kubectl get pods -l app=nginx
```

- **Use Case:** Filter resources based on specific labels.

15. Create a Resource Quota

```
kubectl create quota my-quota --hard=cpu=10,memory=10Gi,pods=10
```

- **Use Case:** Limit resource usage per namespace.

16. Drain a Node for Maintenance

```
kubectl drain [NODE_NAME] --ignore-daemonsets
```

- **Use Case:** Safely evacuate all pods from a node for maintenance.

17. Watch Resource Changes in Real-Time

```
kubectl get pods --watch
```

- **Use Case:** Monitor updates to pods in real-time.

18. Copy Files From Pod to Local Machine

```
kubectl cp [NAMESPACE]/[POD_NAME]:/path/to/remote/file /path/to/local/file
```

- **Use Case:** Transfer files between pod and local system.

19. Delete Pods Not in a Running State

```
kubectl get pods | grep -v Running | cut -d' ' -f1 | xargs kubectl delete
pod
```

- **Use Case:** Clean up non-running pods to maintain a healthy environment.

20. Port Forward to Local Machine

```
kubectl port-forward [POD_NAME] [LOCAL_PORT]:[REMOTE_PORT]
```

- **Use Case:** Access and manage services from a local machine.

Part 2: Intermediate Kubernetes Commands & Operational Tricks

21. Check Cluster Resource Availability

```
kubectl top nodes
```

- **Use Case:** Monitor the usage of CPU and memory resources across nodes in the cluster.

22. Label a Node for Specific Deployments

```
kubectl label nodes [NODE_NAME] hardware=high-spec
```

- **Use Case:** Assign labels to nodes to target them with specific pods that require higher specifications.

23. Get Detailed Node Information

```
kubectl describe node [NODE_NAME]
```

- **Use Case:** Fetch detailed information about a node, including its status, labels, conditions, and assigned pods.

24. Taint a Node to Control Pod Placement

```
kubectl taint nodes [NODE_NAME] key=value:NoSchedule
```

- **Use Case:** Apply a taint to a node to prevent pods from being scheduled on it unless they tolerate the taint.

25. Patch a Running Pod

```
kubectl patch pod [POD_NAME] -p
'{"spec":{"containers":[{"name":"[CONTAINER_NAME]","image":"[NEW_IMAGE]"}]}
}'
```

- **Use Case:** Update a specific aspect of a running pod, such as the container image.

26. Decode Secrets

```
kubectl get secret [SECRET_NAME] -o jsonpath="{.data.token}" | base64 --
decode
```

- **Use Case:** Decode and view Kubernetes secrets, which are stored encoded by default.

27. Export Current State of Resources to a File

```
kubectl get deployment [DEPLOYMENT_NAME] -o yaml --export > deployment.yaml
```

- **Use Case:** Save the current state of a deployment or any other resource to a YAML file for backup or replication.

28. Restart a Deployment

```
kubectl rollout restart deployment/[DEPLOYMENT_NAME]
```

- **Use Case:** Restart all pods in a deployment, useful for refreshing the application without changing the deployment configuration.

29. Use JSON Path Queries for Custom Outputs

```
kubectl get pods -o=jsonpath='{range
.items[*]}{.metadata.name}{"\t"}{.status.phase}{"\n"}{end}'
```

- **Use Case:** Customize the output of `kubectl` commands to display specific data fields in a specified format.

30. Change Namespace for the Current Context

```
kubectl config set-context --current --namespace=[NAMESPACE]
```

- **Use Case:** Switch the default namespace of the current context, simplifying commands that follow.

31. Simplify Complex Kubernetes YAML with Kustomize

- **Usage:**

```
kubectl apply -k [KUSTOMIZATION_DIRECTORY]
```

- **Use Case:** Manage application configuration with Kustomize, which allows for template-free customization of multiple Kubernetes manifests.

32. Monitor Pod Disruption Budgets

```
kubectl get poddisruptionbudgets
```

- **Use Case:** Ensure that the minimum number of replicas of an application remain available during voluntary disruptions.

33. Schedule Jobs for Specific Times

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: example-cronjob
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: example-container
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

- **Use Case:** Run batch jobs at specific times using CronJob resources.

34. Interactively Manage and Debug Pods

```
kubectl run -i --tty busybox --image=busybox -- sh
```

- **Use Case:** Launch an interactive shell session within a pod for troubleshooting and debugging.

35. Analyze and Debug Network Policies

```
kubectl run --generator=run-pod/v1 tmp-shell --rm -it --image
nicolaka/netshoot -- bash
```

- **Use Case:** Use a temporary pod with network troubleshooting tools to test and debug network policies.

36. View Evicted Pods

```
kubectl get pods --field-selector=status.phase=Failed
```

- **Use Case:** Identify pods that have been evicted due to resource constraints or node failures.

37. Automate Service Exposure

```
kubectl expose deployment [DEPLOYMENT_NAME] --port=[PORT] --
type=LoadBalancer
```

- **Use Case:** Quickly create a service that exposes a deployment externally, allocating a public IP if on a supported cloud provider.

38. Force Delete Pods in Terminating State

```
kubectl delete pods [POD_NAME] --grace-period=0 --force
```

- **Use Case:** Forcefully delete pods that are stuck in a terminating state, which can occur due to various issues.

39. Backup and Restore Etcd

- **Backup:**

```
ETCDCTL_API=3 etcdctl snapshot save snapshot.db
```

- **Restore:**

```
ETCDCTL_API=3 etcdctl snapshot restore snapshot.db
```

- **Use Case:** Safeguard and recover the Kubernetes cluster's state by backing up and restoring the Etcd datastore.

40. Aggregate Logs Using Stern

- **Command:**

```
stern [POD_NAME_PATTERN] --since 1h
```

- **Use Case:** Tail logs from multiple pods matching the name pattern, useful for debugging applications spanning multiple pods.

Part 3: Advanced Kubernetes Commands & Performance Tricks

41. Use Node Affinity to Control Pod Placement

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  containers:
  - name: with-node-affinity
    image: k8s.gcr.io/pause:2.0
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
```

```
            - e2e-az1
            - e2e-az2
```

- **Use Case:** Ensure pods are scheduled on nodes in specific availability zones, enhancing performance and reliability.

42. Horizontal Pod Autoscaler Based on Custom Metrics

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: custom-metric-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Pods
    pods:
      metric:
        name: packets-processed
      target:
        type: AverageValue
        averageValue: 1000
```

- **Use Case:** Scale applications dynamically based on custom metrics like processed packets, optimizing resource use and application responsiveness.

43. Optimize Cluster Scheduling with Pod Priority and Preemption

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used for XYZ service pods
only."
---
apiVersion: v1
kind: Pod
metadata:
  name: high-priority-pod
spec:
  containers:
  - name: high-priority
    image: nginx
  priorityClassName: high-priority
```

- **Use Case:** Prioritize critical service pods over others, ensuring they are scheduled and run preferentially.

44. Isolate Namespaces Using Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: restricted
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

- **Use Case:** Enhance security by default denying all ingress and egress traffic in sensitive namespaces, requiring explicit allowances.

45. Monitor API Server Requests

```
kubectl top --sort-by=cpu
```

- **Use Case:** Identify which components or services are consuming the most CPU resources on the API server, helping in diagnosing performance issues.

46. Graceful Pod Shutdown with PreStop Hook

```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo
    image: nginx
    lifecycle:
      preStop:
        exec:
          command: ["/usr/sbin/nginx","-s","quit"]
```

- **Use Case:** Ensure that services handle termination signals gracefully, allowing them to finish critical tasks before shutdown.

47. Optimize Persistent Volume Usage with StorageClasses

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

- **Use Case:** Define storage classes with different performance characteristics, such as SSDs for high-throughput applications.

48. Advanced Logging with Fluentd

- **Setup Fluentd to Aggregate and Forward Logs:**

```
kubectl create -f https://k8s.io/examples/debug/fluentd-daemonset.yaml
```

- **Use Case:** Collect logs from all nodes and pods, forwarding them to a central logging service like Elasticsearch for more sophisticated analysis.

49. Control Pod Egress Traffic Using Egress Gateway

- **Implement an Egress Gateway:**

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-egress
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: gateway
  egress:
  - to:
    - ipBlock:
        cidr: 1.2.3.4/32
```

-

Use Case: Regulate and monitor outbound traffic from your cluster to meet compliance and security requirements.

50. Use Init Containers for Setup Scripts

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
spec:
  containers:
  - name: myapp-container
    image: myapp
  initContainers:
  - name: init-myservice
    image: busybox
    command: ['sh', '-c', 'echo initializing && sleep 1']
```

- **Use Case:** Execute preliminary setup tasks before the main application starts, ensuring that all dependencies or prerequisites are met.

Part 4: Advanced Kubernetes Commands & Cluster Management Tricks

51. Automatically Replace Unhealthy Nodes

```
kubectl get nodes --no-headers | awk '{if ($2 != "Ready") print $1}' |
xargs kubectl delete node
```

- **Use Case:** Automatically identify and remove nodes that are not in the 'Ready' state, ensuring the cluster's health and reliability.

52. Set Resource Limits for Namespace

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-quota
  namespace: prod
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

- **Use Case:** Enforce specific CPU and memory limits at the namespace level to prevent any one project from consuming excessive cluster resources.

53. Use Custom Scheduler for Specialized Workloads

```
apiVersion: v1
kind: Pod
metadata:
  name: with-custom-scheduler
spec:
  schedulerName: my-scheduler
  containers:
  - name: nginx
    image: nginx
```

- **Use Case:** Deploy pods using a custom scheduler tailored to specific needs or optimizations, instead of the default Kubernetes scheduler.

54. Manage Cluster Access with Role-Based Access Control (RBAC)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

- **Use Case:** Secure your Kubernetes environment by specifying who can access which resources, ensuring users only have the necessary permissions.

55. Advanced Pod Scheduling with Affinity and Anti-Affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: "kubernetes.io/hostname"
  containers:
  - name: myapp
    image: myapp
```

- **Use Case:** Place pods based on the labels of other pods and the nodes they are located on, enhancing the co-location or distribution of workloads across the cluster.

56. Encrypt Secrets at Rest

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

- **Configuration:**

```
# Enable encryption at rest in Kubernetes by configuring the API server
# with a proper encryption provider configuration.
```

- **Use Case:** Protect sensitive data, ensuring that secrets like passwords or API keys are encrypted in storage, not just in transit.

57. Configure Automatic Sidecar Injection with Istio

- **Enable Istio on a Namespace:**

```
kubectl label namespace default istio-injection=enabled
```

- **Use Case:** Automatically add an Istio sidecar proxy to eligible pods within a namespace to secure and manage network traffic.

58. Monitor Resource Usage in Real-Time

```
kubectl top pod --containers
```

- **Use Case:** Display current CPU and memory usage for each container in a pod, helping in quick diagnostics and resource management.

59. Automate Certificate Management with Cert-Manager

- **Install Cert-Manager:**

```
kubectl create namespace cert-manager
kubectl apply --validate=false -f https://github.com/jetstack/cert-
manager/releases/download/v1.0.4/cert-manager.yaml
```

- **Use Case:** Automate the issuance and renewal of SSL/TLS certificates, ensuring secure communication within the cluster.

60. Efficiently Handle Node Evacuation

```
kubectl drain [NODE_NAME] --ignore-daemonsets --delete-emptydir-data
```

- **Use Case:** Safely evacuate all pods from a node while respecting Kubernetes' data management policies, ideal for performing maintenance or upgrades without data loss.

61. Leverage Kubernetes Events for Troubleshooting

```
kubectl get events --sort-by='.metadata.creationTimestamp'
```

- **Use Case:** Gather a chronological order of cluster events to troubleshoot and understand

the sequence of actions that have affected the cluster.

62. Configure Pod Disruption Budgets for High Availability

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: myapp
```

- **Use Case:** Protect critical applications from voluntary disruptions during maintenance, ensuring at least a certain number of replicas remain running.

63. Utilize Vertical Pod Autoscaling

- **Enable VPA:**

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/autoscaler/master/vertical-
pod-autoscaler/deploy/recommended.yaml
```

- **Use Case:** Automatically adjust the CPU and memory reservations of pods based on their usage, optimizing resource allocation.

64. Implement Network Policies for Pod Security

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: restrict-internal-traffic
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

- **Use Case:** Define rules that restrict how pods communicate with each other, improving the security posture of your environment by preventing unauthorized access.

65. Backup Kubernetes Cluster Data Using Velero

- **Install Velero:**

```
velero install --provider aws --bucket my-backup --secret-file
./credentials-velero
```

- **Use Case:** Provide comprehensive backups of cluster resources and persistent volumes, enabling disaster recovery strategies.

Part 5: Advanced Kubernetes Commands & Security Enhancements

66. Secure Pod-to-Pod Communication with Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
```

```
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

- **Use Case:** Establish a default deny all network posture, forcing all traffic to be explicitly permitted, which enhances the security of pod communications.

67. Automated Security Scans with Kube-Bench

- **Command:**

```
kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml
```

- **Use Case:** Run the CIS Kubernetes Benchmark to check for dozens of common best-practices around deploying Kubernetes securely.

68. Use Helm for Managing Complex Deployments

```
helm install my-release stable/my-chart
```

- **Use Case:** Simplify the deployment and management of complex Kubernetes applications with Helm charts, providing templating, versioning, and release management.

69. Implement Readiness and Liveness Probes

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: myimage
    readinessProbe:
      httpGet:
        path: /healthz
        port: 8080
    livenessProbe:
      tcpSocket:
        port: 8080
```

- **Use Case:** Ensure that your applications are running smoothly and are accessible with probes that check the health of containers.

70. Optimize Cluster Logs with EFK Stack (Elasticsearch, Fluentd, Kibana)

- **Setup:**

```
kubectl apply -f https://github.com/elastic/cloud-on-
k8s/tree/master/config/samples
```

- **Use Case:** Centralize logging across the cluster for better insights and debugging capabilities using a robust logging stack.

71. Automate Database Backups with CronJobs

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: db-backup
spec:
  schedule: "0 2 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: db-backup
            image: mydbbackup:latest
            env:
            - name: DB_HOST
              value: mydbhost
```

- **Use Case:** Schedule regular database backups to ensure data durability and recoverability.

72. Isolate Sensitive Workloads with Service Mesh (Istio)

- **Setup Istio:**

```
istioctl install --set profile=demo
```

- **Use Case:** Enhance security, observability, and traffic management of services with a service mesh that offers fine-grained control and encryption.

73. Manage Cluster Secrets with HashiCorp Vault

- **Integration:**

```
vault operator init
```

- **Use Case:** Secure, store, and tightly control access to tokens, passwords, certificates, and other secrets in modern computing environments.

74. Scale Stateful Applications with StatefulSets

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mystatefulapp
```

```
spec:
  serviceName: "mystatefulservice"
  replicas: 3
  selector:
    matchLabels:
      app: mystatefulapp
  template:
    metadata:
      labels:
        app: mystatefulapp
    spec:
      containers:
      - name: mycontainer
        image: myimage
```

- **Use Case:** Manage stateful applications with stable, unique network identifiers, stable persistent storage, and ordered deployment and scaling.

75. Dynamic Configuration with ConfigMaps and Secrets

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfig
data:
  config.json: |
    {"key":"value"}
---
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: MWYyZDFlMmU2N2Rm
```

- **Use Case:** Manage configuration data and sensitive information separately from the pod specification, allowing for easier application configuration and security.

76. Enhance Node Security with Pod Security Policies

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
spec:
  privileged: false


  allowPrivilegeEscalation: false
```

- **Use Case:** Enforce security-related policies like preventing privileged containers, which can be crucial for maintaining the security posture of your cluster.

77. Zero-downtime Deployments with Rolling Updates

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: mycontainer
        image: myimage:v2
```

- **Use Case:** Update applications with no downtime by ensuring that the new version is tested and rolled out gradually while the old version is still running.

78. Track Resource Usage with Metrics Server

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
server/releases/latest/download/components.yaml
```

- **Use Case:** Monitor resource usage of nodes and pods in your cluster, enabling auto-scaling and more informed resource allocation decisions.

79. Use Kubernetes Dashboard for Cluster Management

- **Setup Dashboard:**

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/re
commended.yaml
```

- **Use Case:** Access an intuitive web-based user interface for managing and troubleshooting Kubernetes cluster resources.

80. Customize Pod Scheduling with Taints and Tolerations

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: myimage
  tolerations:
  - key: "app"
    operator: "Exists"
    effect: "NoSchedule"
```

- **Use Case:** Ensure that certain nodes only accept specific types of pods, allowing you to segregate workloads based on requirements like performance or security.

Part 7: Enhanced Operational Strategies in Kubernetes

91. Automate Security Patching with Kured

- **Install Kured:**

```
kubectl apply -f
https://github.com/weaveworks/kured/releases/download/1.6.1/kured-
1.6.1.yaml
```

- **Use Case:** Use Kured (KUbernetes REboot Daemon) to safely automate node reboots after security updates, minimizing manual intervention and maintaining security compliance.

92. Data Encryption in Transit with mTLS

- **Configure Istio for mTLS:**

```
istioctl install --set values.global.mtls.enabled=true
kubectl label namespace default istio-injection=enabled
```

- **Use Case:** Secure all data in transit within the cluster by enabling mutual TLS, ensuring that all communications between services are encrypted.

93. Backup and Restore Cluster Data with Stash

- **Install Stash:**

```
kubectl apply -f https://raw.githubusercontent.com/stashed/stash/v0.9.0-
rc.2/deploy/kubernetes.yaml
```

- **Use Case:** Implement a robust backup and recovery strategy for Kubernetes resources and persistent volumes, ensuring data integrity and availability.

94. Integrate OPA (Open Policy Agent) for Policy Enforcement

```
kubectl create namespace opa
kubectl apply -f https://raw.githubusercontent.com/open-policy-
agent/opa/main/quick_start.yaml
```

- **Use Case:** Enforce fine-grained, context-aware policies across the Kubernetes stack to maintain compliance and governance standards.

95. Optimize Cost with Spot Instances

- **Configure Spot Instances:**

```
apiVersion: autoscaling.k8s.io/v1
kind: ClusterAutoscaler
metadata:
  name: cluster-autoscaler
spec:
  behavior:
    scaleDown:
      enabled: true
      utilizationThreshold: 0.5
```

- **Use Case:** Leverage spot instances for non-critical workloads to significantly reduce costs without compromising performance.

96. Implement GitOps for Kubernetes Management

- **Setup ArgoCD:**

```
kubectl create namespace argocd
kubectl apply - n argocd -f
https://raw.githubusercontent.com/argoproj/argo-
cd/stable/manifests/install.yaml
```

- **Use Case:** Manage and synchronize Kubernetes resources directly from Git repositories, enabling version-controlled and declarative infrastructure.

97. Enhance Logging with Loki

- **Install Loki:**

```
kubectl apply -f
https://raw.githubusercontent.com/grafana/loki/main/production/ksonnet/loki
/install.yaml
```

- **Use Case:** Implement a horizontally scalable, highly available, multi-tenant log aggregation system based on the same design as Prometheus.

98. Cluster Federation for Global Load Balancing

- **Setup Cluster Federation:**

```
kubefedctl federate enable clusters
```

- **Use Case:** Manage multiple Kubernetes clusters as a single entity with unified deployment and scaling strategies, improving global availability and load distribution.

99. Track Configuration Drift with Datree

- **Integrate Datree:**

```
kubectl apply -f
https://raw.githubusercontent.com/datreeio/datree/blame/main/installation.y
aml
```

- **Use Case:** Automatically check Kubernetes configurations against best practices to prevent drift and ensure consistent configurations across deployments.

100. Leverage Service Mesh for Advanced Traffic Management

- **Configure Advanced Istio Features:**

```
istioctl install --set profile=default --set
values.telemetry.v2.enabled=true
```

- **Use Case:** Utilize service mesh capabilities to control traffic flow, implement advanced routing rules, and gain in-depth telemetry insights.