DevOps Shack

Kubernetes Tips & Useful Tricks With Usecases | Part-1,2,3

[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.