Kubernetes Trouble Cheat Sheet

Most helpful debugging commands

Onboard new cluster	
Set multiple kubeconfig files	export KUBECONFIG=~/.kube/conf1:~/.kube/conf2:~/.kube/conf3
Merge multiple kubeconfig into one file	export KUBECONFIG=xxx kubectl config view --flatten > ~/.kube/merged-config
Set user credentials	kubectl config set-credentials <user-name>
Display current kubeconfig	kubectl config view
List contexts, find the new cluster	kubectl config get-contexts
Switch to a specific context	kubectl config use-context <cluster name>
Check connection and Versions	kubectl version // outdated?
Show Cluster Info	kubectl cluster-info
What namespaces can I access	kubectl get namespaces
Set default namespace	kubectl config set-context --current --namespace=production
Let's look for trouble	
Show recent events	kubectl get events -A --sort-by=.metadata.creationTimestamp
Show node status & version	kubectl get nodes -o wide // all nodes ready?
Show node resource usage	kubectl top nodes // is a node super busy?
Show node details	kubectl describe node <node_name>
Show restarting pods	kubectl get pods -o wide -A --sort-by='.status.containerStatuses[0].restartCount'
Show busy pods	kubectl top pods --all-namespaces --sort-by=cpu
Logs from all pods with label	kubectl logs --tail=500 -l app=xxxx -f // use with | grep to filter
Turn it off and on again	
Redeploy all pods of a deployment	kubectl rollout restart deployment/myapp
Rescale deployments	kubectl scale --current-replicas=2 --replicas=3 deployment/mysql
Delete stuck pods	kubectl delete pod unwanted --now
We need to dig deeper	
Launch a debug container	kubectl debug -it --container=debugger --image=alpine --target=namespace pod
Forward a port to debug against	kubectl port-forward <pod name> <portliste>:<portforwar>
Execute a command in a pod	kubectl exec -it <pod-name> -- /bin/bash
Copy files over from pod to local	kubectl cp <pod-name>:<path-to-file> <path-to-local-file> -c <container-name>
Copy files over from local to pod	kubectl cp <path-to-local-file> <pod-name>:<path-to-file>
A node in trouble	
Mark node as unschedulable	kubectl cordon node <node_name>
Mark node as schedulable	kubectl uncordon node <node_name>
Evict all pods from a node	kubectl drain node <node_name>
Apply a taint to a node	kubectl taint node <node_name>
Add or modify annotations of a node	kubectl annotate node <node_name>

Best Practices - Low hanging fruits

1. Use namespaces

Use a namespace for each logical partition to enable better access control with RBAC, apply network policies, prioritize computation resources with Resource Quotas, and simplify administration tasks.

2. Use declarative configuration with GitOps

No direct edits with kubectl beyond troubleshooting! Use **$ kubectl apply -f my-conf.yaml** and make changes through YAML configurations to ensure reproducible deployments. Store these in version control. **Even better: use a CI/CD pipeline** to apply changes and give developers read-only permissions.

3. Use readiness and liveness probes

Without a readiness probe, a container will receive traffic right after launch, even if it might not be ready. Liveness probes check if the app is still alive and restart the container if it is not.

```
spec.containers:
  livenessProbe:
    httpGet:
      path: /healthcheck
      port: 8000
  readinessProbe:
    httpGet:
      path: /healthcheck
      port: 8000
```

4. Don't run as root & immutable pods

Don't use the local filesystem to store state—containers should be stateless. Make pods immutable with **'readOnlyRootFilesystem'** and use **'emptyDir'** volumes if file system writes are necessary. Don't run processes with root privileges and block privilege escalation. This will significantly reduce the attack surface.

```
spec.containers:
    securityContext:
      allowPrivilegeEscalation: false
      privileged: false
      readOnlyRootFilesystem: true
      runAsGroup: 101
      runAsUser: 101
```

5. Reduce containers size and update base image regularly

Aim for **small container images**! This accelerates builds and deployments and significantly reduces the attack surface. Alpine is a good starting point for most applications. Make sure to keep the base image up to date.

6. Containers should crash on error

Don't handle errors or exceptions inside the containers. Instead, **let them crash and exit.** This allows kubelet to restart the service and provides better observability of app states at a cluster level. **Exception: don't crash** if a dependency (e.g., a database) isn't ready; retry instead to avoid a CrashLoopBackOff.

7. Use resource requests and limits

Don't let a single process block or OOM a node.
Use **cpu requests and memory requests and limits**. **Don't use limit cpu**. KRR (Kubernetes Resource Recommender) is a great tool to help setting good limits. Containers without limits are treated as low priority and will be evicted first in case of scarce resources.

```
spec.containers.resources:
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
```

8. Spread pods with affinity

Pods of a specific app should not be grouped on one node; if that node fails, the entire app could go down. Add an affinity rule to distribute the pods across multiple nodes using a suitable label.

```
spec.topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: kubernetes.io/hostname
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        app: "my-app"
```

7. Use Pod disruption budgets

Busy services may need to maintain a certain number of pods. To protect a service from unexpected events that could take down several pods simultaneously, define a Pod Disruption Budget.

```
spec:
    minAvailable: 50%
```

8. Have a plan for secrets

Obviously, you shouldn't have clear text secrets in git. Not having a storage solution violates rule #2. A simple approach is to use sealed-secrets. This will encrypt secrets with a known public key, allowing you to commit secrets to git. Even better, use an external service to manage your secrets in a vault.

Sven.Eliasson@gmail.com Fullstack / DevOps - Kubernetes / Clickhouse 18.05.2024, 1.0