Kubernetes (K8s) Essentials Cheatsheet

A practical and concise Kubernetes (K8s) cheatsheet designed for DevOps engineers and SREs, covering essential commands, concepts, and best practices for managing containerized applications.

Kubernetes Core Concepts & Operations

KUBERNETES BASICS

`kubectl` Syntax	`kubectl <command> <type> <name> [flags]` **Example:** `kubectl get pods -n my-app`
Get Resources	`kubectl get <resource-type>` List pods: `kubectl get pods` List all resources: `kubectl get all` Wide output: `kubectl get pods -o wide`
Describe Resources	`kubectl describe <resource-type> <name>` Detailed info for a pod: `kubectl describe pod my-pod-xyz` Crucial for debugging.
Context & Config Switching	List contexts: `kubectl config get-contexts` Switch context: `kubectl config use-context <context-name>` **Tip:** Use `KUBECONFIG` env var for specific configs.
Namespace Operations	List namespaces: `kubectl get ns` Switch namespace: `kubectl config set-context --current --namespace=<ns-name>` Specify namespace with `-n` or `--namespace` flag: `kubectl get pods -n kube-system`
Current Context/Namespace	Show current context: `kubectl config current-context` Show current namespace: `kubectl config view --minify | grep namespace:`
Pro Tip / Gotcha	**Gotcha:** Always check your current context and namespace before running commands that modify resources, especially in production environments. A simple `kubectl config current-context` can save you headaches.

PODS & DEPLOYMENTS

Creating/Applying Resources	`kubectl apply -f <file.yaml>` Creates or updates resources based on YAML definition. **Example:** `kubectl apply -f my-deployment.yaml`

Listing Pods/Deployments	`kubectl get pods`
	`kubectl get deployments`
	
	Use `-w` or `--watch` to stream updates: `kubectl get pods -w`

Deleting Resources	`kubectl delete <resource-type> <name>`
	`kubectl delete -f <file.yaml>`
	
	Caution: `--force --grace-period=0` can forcefully delete but may lead to data loss or orphaned resources. Use with care.

Pod vs. Deployment YAML	**Pod:** Basic unit, ephemeral. `apiVersion: v1`, `kind: Pod` **Deployment:** Manages Pods, provides declarative updates. `apiVersion: apps/v1`, `kind: Deployment` **Best Practice:** Always use Deployments for stateless apps for easy scaling and rolling updates.
Scaling Deployments	`kubectl scale deployment <name> --replicas=<count>` **Example:** `kubectl scale deployment my-app --replicas=3`
Rolling Updates	Update image: `kubectl set image deployment/<name> <container>=<new-image>:<tag>` **Example:** `kubectl set image deployment/nginx-deployment nginx=nginx:1.19.0` Rollback: `kubectl rollout undo deployment/<name>`
Pro Tip / Gotcha	**Pro Tip:** Use `kubectl rollout status deployment/<name>` to monitor the progress of your rolling updates. This is crucial for verifying successful deployments or quickly spotting issues.

SERVICES & NETWORKING

Service Types: ClusterIP	Default type, exposes service on an internal IP. Only reachable from within the cluster. **Use for:** Internal services, backend components.
Service Types: NodePort	Exposes service on a static port on each Node's IP. Accessible from outside the cluster via `<NodeIP>:<NodePort>`. **Use for:** Exposing services on test clusters or when a LoadBalancer isn't available.
Service Types: LoadBalancer	Exposes service externally using a cloud provider's load balancer. Get an external IP. **Use for:** Production-grade external access to web applications.
Ingress Basics	Manages external access to services within a cluster, typically HTTP/S. Requires an Ingress Controller (e.g., Nginx, Traefik). Defines routing rules based on host/path.
DNS Resolution	Services are discoverable via DNS: `<service-name>` -> `<ClusterIP>` (within same namespace) `<service-name>.<namespace>.svc.cluster.local` -> `<ClusterIP>` (across namespaces) **Example:** `ping my-service` or `curl http://my-service.default.svc.cluster.local`
Port Forwarding	`kubectl port-forward <pod-name> <local-port>:<container-port>` **Example:** `kubectl port-forward my-app-pod 8080:80` Access local: `http://localhost:8080` to connect to `my-app-pod`'s port 80.
Pro Tip / Gotcha	**Gotcha:** NodePort exposes your service on *all* nodes, potentially including those you don't expect traffic on. For production, LoadBalancer or Ingress is almost always preferred for better security and routing.

K8s Configuration, Storage, & Security

CONFIGMAPS & SECRETS

Creating ConfigMaps (Literal)	`kubectl create configmap <name> --from-literal=<key>=<value>` **Example:** `kubectl create configmap app-config --from-literal=DB_HOST=mysql --from-literal=APP_PORT=8080`
Creating ConfigMaps (File)	`kubectl create configmap <name> --from-file=<file-path>` **Example:** `kubectl create configmap my-nginx-config --from-file=nginx.conf` **Multiple files:** `--from-file=dir/`
Creating Secrets (Literal)	`kubectl create secret generic <name> --from-literal=<key>=<value>` **Example:** `kubectl create secret generic db-creds --from-literal=username=admin --from-literal=password=supersecret`
Creating Secrets (File)	`kubectl create secret generic <name> --from-file=<file-path>` **Example:** `kubectl create secret generic tls-certs --from-file=tls.crt --from-file=tls.key`
Mounting into Pods	**As Env Vars:** `envFrom: configMapRef: name: my-config` **As Files:** `volumeMounts` and `volumes` in pod spec. Mounted as read-only files.
Differences & Security	**ConfigMaps:** Store non-confidential data (plain text). **Secrets:** Store sensitive data (base64 encoded, not encrypted by default on etcd). **Security Tip:** Encrypt Secrets at rest using KMS or tools like `Sealed Secrets` for production.
Pro Tip / Gotcha	**Gotcha:** `kubectl get secret <name> -o yaml` will show the base64-encoded value. To decode, use `echo <value> | base64 --decode`. Never store raw sensitive data in Git.

VOLUMES & STORAGE

EmptyDir	A temporary, empty directory created when a Pod is assigned to a node. Deleted when the Pod is removed from the node. **Use for:** Scratch space, caching, temporary file storage.
hostPath	Mounts a file or directory from the host node's filesystem into a Pod. **Caution:** Not recommended for most uses due to security and scheduling issues. Ties Pod to a specific node.
PersistentVolumeClaim (PVC)	A request for storage by a user. Consumes PV resources. Namespace-scoped. **YAML:** `kind: PersistentVolumeClaim`, `accessModes`, `resources: requests`.
PersistentVolume (PV)	A piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned. Cluster-scoped. **YAML:** `kind: PersistentVolume`, `capacity`, `accessModes`, `storageClassName`, `persistentVolumeReclaimPolicy`.
StorageClass Usage	Defines "classes" of storage. Allows dynamic provisioning of PVs when a PVC requests a specific StorageClass. **Example:** `storageClassName: standard` or `ssd`.
Deleting PVC/PV	Delete PVC: `kubectl delete pvc <name>` This will trigger PV deletion if `reclaimPolicy: Delete`. If `reclaimPolicy: Retain`, the PV remains (in 'Released' state) and must be manually deleted. Underlying storage might persist.
Pro Tip / Gotcha	**Pro Tip:** For stateful applications, always use PVCs, which abstract the underlying storage. This makes your applications portable and less coupled to specific infrastructure.

MONITORING & DEBUGGING

Viewing Logs (`kubectl logs`)	Get logs: `kubectl logs <pod-name>` Follow logs: `kubectl logs -f <pod-name>` Previous container: `kubectl logs -p <pod-name>` Specific container: `kubectl logs <pod-name> -c <container-name>`
Executing Commands (`kubectl exec`)	Run command: `kubectl exec <pod-name> -- ls -l /app` Interactive shell: `kubectl exec -it <pod-name> -- /bin/bash` (or `/bin/sh`) **Use for:** Quick debugging, file inspection inside a running container.
Describing Resources (`kubectl describe`)	`kubectl describe <resource-type> <name>` Provides a detailed status, events, and configuration. Essential for understanding why a pod isn't starting or behaving as expected.
Events	Check events: `kubectl get events` or `kubectl describe <resource>` Events show what happened to a resource (e.g., Pod scheduled, Container pulled, Failed). Look for `Warning` or `Error` types.
Liveness/Readiness Probes	**Liveness Probe:** Checks if app is running. If fails, K8s restarts container. **Readiness Probe:** Checks if app is ready to serve traffic. If fails, Pod removed from Service endpoints. **Best Practice:** Always define these for production apps.

Resource Usage (`kubectl top`)	Requires Metrics Server deployment in cluster.
	
	`kubectl top pods`
	`kubectl top nodes`
	
	Use for: Quickly checking CPU/Memory usage of pods and nodes.

Pro Tip / Gotcha	**Gotcha:** If your `kubectl logs` command isn't showing anything, double-check your container name with `kubectl describe pod <pod-name>` and ensure the container is actually running.

RBAC & SECURITY

ServiceAccounts	Provides an identity for processes that run in a Pod. Pods typically run under a default ServiceAccount in their namespace. Can be explicitly assigned via `serviceAccountName` in pod spec.
Roles & ClusterRoles	**Role:** Grants permissions within a specific namespace. `kind: Role` **ClusterRole:** Grants permissions across all namespaces or for cluster-scoped resources. `kind: ClusterRole` Defines `apiGroups`, `resources`, `verbs` (get, list, create, delete, etc.).
RoleBindings & ClusterRoleBindings	**RoleBinding:** Binds a Role (or ClusterRole) to a ServiceAccount, User, or Group within a namespace. `kind: RoleBinding` **ClusterRoleBinding:** Binds a ClusterRole to a ServiceAccount, User, or Group across the cluster. `kind: ClusterRoleBinding`
Viewing RBAC	`kubectl get role,rolebinding,clusterrole,clusterrolebinding -A` Test user permissions: `kubectl auth can-i <verb> <resource> --as=<user> -n <namespace>` **Example:** `kubectl auth can-i get pods --as=dev-user -n my-app`
Network Policies (Ingress)	Controls inbound traffic to Pods. **Default:** All pods are non-isolated and accept all traffic. Once a NetworkPolicy selects a Pod, it becomes isolated and rejects traffic not explicitly allowed.
Network Policies (Egress)	Controls outbound traffic from Pods. **Default:** All pods allow all outbound traffic. Once a NetworkPolicy selects a Pod, its egress traffic is restricted to only what's explicitly allowed.
Pro Tip / Gotcha	**Pro Tip:** Implement `least privilege` principle for ServiceAccounts and RBAC. Grant only the necessary permissions to prevent security vulnerabilities. Start with minimal permissions and add more as needed.