Kubernetes YAML Blueprint Bible — Admin + Developer (Full verbose, cheat-sheet style)

This PDF contains YAML file examples (with line-by-line comments) for cluster admins and developers. Files are ordered and named as requested. Use these as production-ready starting points and adapt to your environment.

1. Cluster-level setup YAMLs

File: cluster.yaml

Purpose: Control-plane bootstrap config for kubeadm

```
# cluster.yaml - kubeadm cluster configuration (control-plane + networking)
# Use with: kubeadm init --config cluster.yaml
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
# Kubernetes version to install
kubernetesVersion: v1.27.3
# Control plane endpoint (load balancer or IP:port)
controlPlaneEndpoint: "loadbalancer.example.com:6443"
networking:
  # Pod network CIDR used by your CNI (change for your CNI provider)
  podSubnet: "10.244.0.0/16"    # e.g., Flannel
  serviceSubnet: "10.96.0.0/12"
apiServer:
  extraArgs:
    authorization-mode: "Node,RBAC"
controllerManager: {}
scheduler: {}
---
apiVersion: kubeadm.k8s.io/v1beta3
kind: InitConfiguration
nodeRegistration:
  name: "master-1"
  criSocket: /var/run/dockershim.sock
  taints:
  - key: "node-role.kubernetes.io/master"
    effect: "NoSchedule"
```

2. Namespace & core objects

File: namespace.yaml

Purpose: Create a logical partition in the cluster

```
# namespace.yaml - create a namespace for isolation
apiVersion: v1
kind: Namespace
metadata:
  name: "production"        # change to your environment (staging, dev, etc.)
  labels:
    env: "production"       # label used for selectors and policies
```

File: serviceaccount.yaml

Purpose: Identity for pods to access the API

```
# serviceaccount.yaml - a service account for pods to use
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app-sa               # name of the service account
  namespace: production      # namespace where it will live
  labels:
    app: "myapp"
```

3. Workload controllers (Pods, Deployments, StatefulSets...)

File: pod.yaml

Purpose: A single Pod with resource requests/limits and env

```
# pod.yaml - single Pod example (not recommended for production long-term)
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-pod
  namespace: production
  labels:
    app: simple
spec:
  serviceAccountName: app-sa   # use the service account defined earlier
  containers:
  - name: nginx
    image: nginx:1.25         # container image and tag
    ports:
    - containerPort: 80       # internal container port
    resources:
      requests:
        cpu: "100m"
        memory: "128Mi"
      limits:
        cpu: "250m"
        memory: "256Mi"
    env:
    - name: ENV
      value: "production"
```

File: deployment.yaml

Purpose: Deployment for stateless web application

```yaml
# deployment.yaml - stateless Deployment managing replicas
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deployment
  namespace: production
  labels:
    app: web
spec:
  replicas: 3                   # number of pod replicas
  selector:
    matchLabels:
      app: web                  # how Deployment finds its pods
  template:
    metadata:
      labels:
        app: web
    spec:
      serviceAccountName: app-sa
      containers:
      - name: web
        image: nginx:1.25       # update image as needed
        ports:
        - containerPort: 80
        livenessProbe:          # simple HTTP liveness check
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:         # readiness to accept traffic
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5
        resources:
          requests:
            cpu: "200m"
            memory: "256Mi"
          limits:
            cpu: "500m"
            memory: "512Mi"
```

File: replicaset.yaml

Purpose: ReplicaSet (Deployment commonly manages this)

```yaml
# replicaset.yaml - rarely created directly (Deployment manages ReplicaSets)
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: web-rs
  namespace: production
  labels:
    app: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web
        image: nginx:1.25
```

File: statefulset.yaml

Purpose: StatefulSet for databases requiring stable identity and storage

```
# statefulset.yaml - stateful app (ordered pod identity + stable storage)
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: db-stateful
  namespace: production
spec:
  serviceName: "db"            # headless service name for stable network IDs
  replicas: 3
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - name: postgres
        image: postgres:15
        ports:
        - containerPort: 5432
        env:
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: password
        volumeMounts:
        - name: data
          mountPath: /var/lib/postgresql/data
  volumeClaimTemplates:        # stable storage per pod
  - metadata:
      name: data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
```

File: daemonset.yaml

Purpose: DaemonSet to run one pod per node for cluster-level services

```
# daemonset.yaml - run a pod on every node (e.g., logging/monitoring agent)
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-agent
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: node-agent
  template:
    metadata:
      labels:
        name: node-agent
    spec:
      tolerations:
      - key: "node-role.kubernetes.io/master"
        operator: "Exists"
        effect: "NoSchedule"
      containers:
      - name: fluentd
```

```
          image: fluent/fluentd:v1.15
          resources:
            limits:
              cpu: "200m"
              memory: "200Mi"
```

File: job.yaml

Purpose: Job that runs to completion and exits

```
# job.yaml - single-run Job for batch processing
apiVersion: batch/v1
kind: Job
metadata:
  name: data-migrate
  namespace: production
spec:
  template:
    spec:
      containers:
      - name: migrator
        image: myorg/migrator:latest
        args: ["--migrate"]
      restartPolicy: OnFailure
  backoffLimit: 4
```

File: cronjob.yaml

Purpose: CronJob for scheduled tasks (daily backups)

```
# cronjob.yaml - scheduled Job (CronJob)
apiVersion: batch/v1
kind: CronJob
metadata:
  name: nightly-backup
  namespace: production
spec:
  schedule: "0 2 * * *"   # run daily at 02:00
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: myorg/backup:latest
            args: ["--upload"]
          restartPolicy: OnFailure
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
```

4. Networking (Services, Ingress, NetworkPolicy)

File: service.yaml

Purpose: Service to expose pods inside the cluster

```
# service.yaml - expose Deployment via ClusterIP (default)
apiVersion: v1
kind: Service
metadata:
  name: web-service
  namespace: production
  labels:
    app: web
spec:
  type: ClusterIP            # ClusterIP, NodePort, LoadBalancer
  selector:
    app: web                 # matches pods with label app:web
  ports:
  - port: 80                 # service port
    targetPort: 80           # container port
    protocol: TCP
```

File: ingress.yaml

Purpose: Ingress resource for external HTTP routing

```
# ingress.yaml - HTTP routing to services (requires Ingress controller)
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: web-ingress
```

```
    namespace: production
    annotations:
      kubernetes.io/ingress.class: "nginx"
  spec:
    rules:
    - host: web.example.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: web-service
              port:
                number: 80
    tls:
    - hosts:
      - web.example.com
      secretName: tls-web-secret     # TLS secret containing certs
```

File: networkpolicy.yaml

Purpose: NetworkPolicy to control pod traffic

```
# networkpolicy.yaml - restrict traffic to/from pods with label app:web
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-only-ingress-from-frontend
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 80
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/8
```

5. Configuration & Secrets

File: configmap.yaml

Purpose: ConfigMap for application settings

```
# configmap.yaml - store non-sensitive configuration
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: production
data:
  LOG_LEVEL: "info"
  FEATURE_FLAG: "enabled"
  app.properties: |
    key1=value1
    key2=value2
```

File: secret.yaml

Purpose: Secret for sensitive data (use sealed-secrets or external vault in prod)

```
# secret.yaml - generic secret (base64 or literal)
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
  namespace: production
type: Opaque
stringData:
  username: "dbuser"           # stringData allows plain text, will be stored base64-encoded
  password: "SuperSecretPassword123"
```

File: resourcequota.yaml

Purpose: ResourceQuota to prevent a namespace from consuming too many cluster resources

```
# resourcequota.yaml - limit resources used in a namespace
apiVersion: v1
kind: ResourceQuota
metadata:
  name: production-quota
  namespace: production
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
    persistentvolumeclaims: "10"
```

File: limitrange.yaml

Purpose: LimitRange to provide default resource requests/limits

```
# limitrange.yaml - set defaults and limits for containers in a namespace
apiVersion: v1
kind: LimitRange
metadata:
  name: production-limits
  namespace: production
spec:
  limits:
  - default:
      cpu: "250m"
      memory: "256Mi"
    defaultRequest:
      cpu: "100m"
      memory: "128Mi"
    type: Container
```

6. Storage (PV, PVC, StorageClass)

File: pv.yaml

Purpose: PersistentVolume mapping to physical storage

```
# pv.yaml - PersistentVolume representing actual storage
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-local-1
spec:
  capacity:
    storage: 100Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-fast
  hostPath:
    path: /mnt/data/pv1      # hostPath only for single-node or dev clusters
```

File: pvc.yaml

Purpose: PVC requesting persistent storage for pods

```
# pvc.yaml - PersistentVolumeClaim requesting storage
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-pvc
  namespace: production
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: local-fast
```

File: storageclass.yaml

Purpose: StorageClass to control dynamic provisioning behaviour

```
# storageclass.yaml - StorageClass for dynamic provisioning (example using hostpath-provisioner)
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-fast
provisioner: kubernetes.io/no-provisioner   # for static PVs; change for CSI driver
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Retain
```

File: volumeattachment.yaml

Purpose: VolumeAttachment used by CSI drivers (auto-generated normally)

```
# volumeattachment.yaml - CSI volume attachment object (usually created by CSI drivers automatically)
apiVersion: storage.k8s.io/v1
kind: VolumeAttachment
metadata:
  name: csi-volume-attach-1
spec:
  attacher: csi.example.com
  nodeName: node-1
  source:
    persistentVolumeName: pv-local-1
```

7. RBAC (Roles, ClusterRoles, Bindings)

File: role.yaml

Purpose: Namespace-scoped permissions

```
# role.yaml - namespace-scoped Role
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
  namespace: production
rules:
- apiGroups: [""]    # "" == core API group
  resources: ["pods", "pods/log"]
  verbs: ["get", "watch", "list"]
```

File: clusterrole.yaml

Purpose: ClusterRole for cluster-wide permissions

```
# clusterrole.yaml - cluster-wide permissions
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin-lite
rules:
- apiGroups: [""]
  resources: ["nodes", "namespaces"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch"]
```

File: rolebinding.yaml

Purpose: Bind a Role to subjects within a namespace

```
# rolebinding.yaml - bind Role to a ServiceAccount in a namespace
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods-binding
  namespace: production
subjects:
- kind: ServiceAccount
  name: app-sa
  namespace: production
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

File: clusterrolebinding.yaml

Purpose: Bind ClusterRole cluster-wide

```
# clusterrolebinding.yaml - bind ClusterRole to a user/group/serviceaccount cluster-wide
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-binding
subjects:
- kind: User
  name: "admin@example.com"
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin-lite
  apiGroup: rbac.authorization.k8s.io
```

8. Scheduling & Priorities

File: affinity.yaml

Purpose: Pod anti-affinity to spread pods across nodes

```
# affinity.yaml - pod affinity/anti-affinity example
apiVersion: v1
kind: Pod
metadata:
  name: aff-pod
  namespace: production
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sleep", "3600"]
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - web
        topologyKey: "kubernetes.io/hostname"   # ensure pods are on different nodes
```

File: tolerations.yaml

Purpose: Tolerations to allow scheduling on tainted nodes

```
# tolerations.yaml - allow pod to run on tainted nodes
apiVersion: v1
kind: Pod
metadata:
  name: tolerant-pod
  namespace: production
spec:
  containers:
  - name: app
    image: nginx
  tolerations:
  - key: "node-role.kubernetes.io/master"
    operator: "Exists"
    effect: "NoSchedule"
```

File: priorityclass.yaml

Purpose: PriorityClass to influence preemption and scheduling order

```
# priorityclass.yaml - assign priority to critical pods
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "High priority for critical system pods"
```

9. Custom Resources (CRD & instances)

File: crd.yaml

Purpose: Define a new custom resource type 'Widget'

```
# crd.yaml - CustomResourceDefinition example (simple)
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: widgets.example.com
spec:
  group: example.com
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              size:
                type: string
  scope: Namespaced
  names:
    plural: widgets
    singular: widget
    kind: Widget
    shortNames:
    - w
```

File: customresource.yaml

Purpose: Instance of a CRD (custom resource)

```
# customresource.yaml - an instance of the CRD defined above
apiVersion: example.com/v1
kind: Widget
metadata:
  name: example-widget
  namespace: production
spec:
  size: "large"
```

Notes & Best Practices

- Use `kubectl apply -f <file.yaml>` to create or update resources idempotently.
- Prefer SealedSecrets or external secret stores over plain Secret YAML files in production.
- Test all YAML in a staging cluster; use `kubectl diff -f <file>` and `kubectl apply --server-dry-run=client -f <file>`.
- Keep resource requests/limits and probes configured for production workloads.
- Replace example hostnames, image tags, storage paths and credentials with environment-specific values.