Some Kubernetes mistakes to avoid

Several categories

Security Mistakes

Configuration Mistakes

Scheduling Mistakes

Observability Mistakes

Cluster Management Mistakes

Application Deployment Mistakes

Not using RBAC

- Implementing RBAC is important to ensure the principles of least privilege
- Several resources
  - rules are defined in Role & ClusterRole
  - Role & ClusterRole are associated to user / group / serviceaccount using RoleBinding & ClusterRoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: development
  name: pod-management
rules:
- apiGroups: ["apps"]
  resources: ["deploy"]
  verbs: ["get", "list", “create”, “delete”]
```

Example of a Role providing Pods management capabilities

Using privileged containers

- A privileged container is similar to a process running directly on the host
- Follow the principle of least privilege by setting minimal necessary permissions to containers

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: overprivileged
spec:
  containers:
  - name: nginx
    image: nginx:1.24
    securityContext:
      privileged: true
```

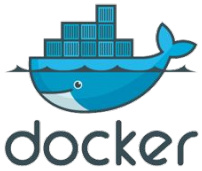
Using container with root privilege

- Configure containers to run as non-root using SecurityContext

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  securityContext:
    runAsUser: 10001
    runAsNonRoot: true
  containers:
  - name: api
    image: registry.gitlab.com/web-hook/api:v1.0.39
```

Not using private container registries

Use private container registries to secure and manage container images as public registries can increase security risks



Not using network segmentation

Use network policies to create segmented network zones within the cluster

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
    matchLabels:
      tiers: db
  policyTypes:
  -   Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tiers: backend
```

Example of a NetworkPolicy which only enables ingress traffic from Pods with a specific label

Not encrypted data in transit

- Use mTLS to encrypt data in transit with a sidecar container
- Might need to consider Service Mesh for this purpose

Not enforcing security policy

- Implement and enforce Pod Security Admission using Pod Security Standard
  - Several types
    - Privileged
    - Restricted
    - Baseline
  - https://kubernetes.io/docs/concepts/security/pod-security-standards/
- OPA Gatekeeper
- Kyverno

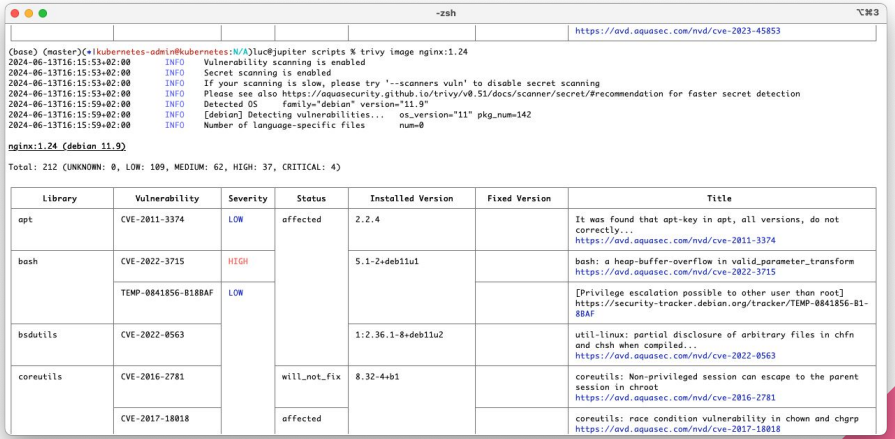
Not using ServiceAccount

- Create and assign specific service accounts with appropriate permissions
- Not all application needs to communicate with the API Server
- Do not mount service account token by default
  - *ServiceAccount.automountServiceAccountToken*
  - *Pod.spec.automountServiceAccountToken*

Not scanning container images for CVEs

Scan container images to limit the risks of CVEs

- Scanning can be done within the CI and regularly in the registry
- Trivy from aquasec



-zsh ⌥⌘3

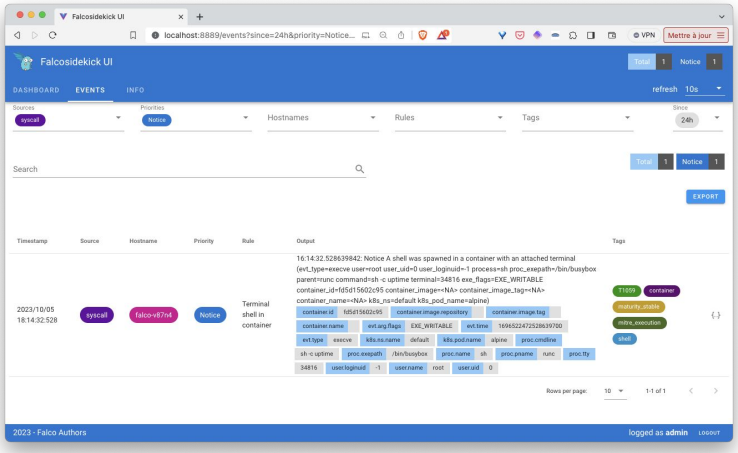
https://avd.aquasec.com/nvd/cve-2023-45853

```
(base) (master)(⎈|kubernetes-admin@kubernetes:N/A)luc@jupiter scripts % trivy image nginx:1.24
2024-06-13T16:15:53+02:00	INFO	Vulnerability scanning is enabled
2024-06-13T16:15:53+02:00	INFO	Secret scanning is enabled
2024-06-13T16:15:53+02:00	INFO	If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2024-06-13T16:15:53+02:00	INFO	Please see also https://aquasecurity.github.io/trivy/v0.51/docs/scanner/secret/#recommendation for faster secret detection
2024-06-13T16:15:59+02:00	INFO	Detected OS	family="debian" version="11.9"
2024-06-13T16:15:59+02:00	INFO	[debian] Detecting vulnerabilities...	os_version="11" pkg_num=142
2024-06-13T16:15:59+02:00	INFO	Number of language-specific files	num=0

nginx:1.24 (debian 11.9)

Total: 212 (UNKNOWN: 0, LOW: 109, MEDIUM: 62, HIGH: 37, CRITICAL: 4)
```

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apt	CVE-2011-3374	LOW	affected	2.2.4		It was found that apt-key in apt, all versions, do not correctly... https://avd.aquasec.com/nvd/cve-2011-3374
bash	CVE-2022-3715	HIGH		5.1-2+deb11u1		bash: a heap-buffer-overflow in valid_parameter_transform https://avd.aquasec.com/nvd/cve-2022-3715
	TEMP-0841856-B18BAF	LOW				[Privilege escalation possible to other user than root] https://security-tracker.debian.org/tracker/TEMP-0841856-B1-8BAF
bsdutils	CVE-2022-0563			1:2.36.1-8+deb11u2		util-linux: partial disclosure of arbitrary files in chfn and chsh when compiled... https://avd.aquasec.com/nvd/cve-2022-0563
coreutils	CVE-2016-2781		will_not_fix	8.32-4+b1		coreutils: Non-privileged session can escape to the parent session in chroot https://avd.aquasec.com/nvd/cve-2016-2781
	CVE-2017-18018		affected			coreutils: race condition vulnerability in chown and chgrp https://avd.aquasec.com/nvd/cve-2017-18018

Not using runtime security tools

Usage of tools like Falco from Sysdig to detect anomalies at runtime



Not using AppArmor

AppArmor defines a profile to restrict access to resources

```
#include <tunables/global>

profile k8s-deny-write flags=(attach_disconnected)
{
  #include <abstractions/base>

  file,

  # Deny all file writes.
  deny /** w,
}
```

```
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
  - name: nginx
    image: nginx
  securityContext:
    appArmorProfile:
      type: Localhost
      localhostProfile: k8s-deny-write
```

Example of an AppArmor profile which blocks all file write operations

Note: before 1.30, the AppArmor profile is set using the Pod's annotation:
container.apparmor.security.beta.kubernetes.io/nginx=localhost/k8s-deny-write

Not configuring a default seccomp profile

Seccomp (Secure Computing Mode) restrict system calls to the underlying Kernel

```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
        "SCMP_ARCH_X86_64",
        "SCMP_ARCH_X86"
    ],
    "syscalls": [
        {
            "names": [
                "accept4",
                …
                "getrlimit"
            ],
            "action": "SCMP_ACT_ALLOW"
        }
    ]
}
```

Example of a Seccomp whitelist profile

```
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
  - name: nginx
    image: nginx
  securityContext:
    seccompProfile:
      type: RuntimeDefault
```

Pod using the seccomp profile of the container runtime

Ignoring compliance standards

Usage of third tools to assess compliance

- kube-bench verifies CIS benchmark
- Kubescape perform checks against several frameworks
  - NSA / MITRE ATT&CK / CIS Benchmark

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
[FAIL] 1.1.9 Ensure that the --profiling argument is set to false (Scored)
[FAIL] 1.1.10 Ensure that the --repair-malformed-updates argument is set to false (Scored)
[PASS] 1.1.11 Ensure that the admission control policy is not set to AlwaysAdmit (Scored)
[FAIL] 1.1.12 Ensure that the admission control policy is set to AlwaysPullImages (Scored)
[FAIL] 1.1.13 Ensure that the admission control policy is set to DenyEscalatingExec (Scored)
[FAIL] 1.1.14 Ensure that the admission control policy is set to SecurityContextDeny (Scored)
[PASS] 1.1.15 Ensure that the admission control policy is set to NamespaceLifecycle (Scored)
[FAIL] 1.1.16 Ensure that the --audit-log-path argument is set as appropriate (Scored)
[FAIL] 1.1.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate (Scored)
[FAIL] 1.1.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as appropriate (Scored)
[FAIL] 1.1.19 Ensure that the --audit-log-maxsize argument is set to 100 or as appropriate (Scored)
[PASS] 1.1.20 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 1.1.21 Ensure that the --token-auth-file parameter is not set (Scored)
[FAIL] 1.1.22 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored)
```

ssh ⌥⌘3

SEVERITY	CONTROL NAME	FAILED RESOURCES	EXCLUDED RESOURCES	ALL RESOURCES	% RISK-SCORE
Critical	Data Destruction	17	0	68	25%
Critical	Disable anonymous access to Kubelet service	0	0	0	skipped**
Critical	Enforce Kubelet client TLS authentication	0	0	0	skipped**
High	CVE-2022-23648-containerd-fs-escape	3	0	3	100%
High	Cluster-admin binding	1	0	68	1%
High	List Kubernetes secrets	10	0	68	15%
High	Privileged container	2	0	12	12%
High	Resources CPU limit and request	10	0	12	74%
High	Resources memory limit and request	10	0	12	74%
High	Workloads with Critical vulnerabilities exposed to external traffic	0	0	0	skipped*
High	Workloads with RCE vulnerabilities exposed to external traffic	0	0	0	skipped*
High	Workloads with excessive amount of vulnerabilities	0	0	0	skipped*
High	Writable hostPath mount	2	0	12	12%
Medium	Access container service account	42	0	42	100%
Medium	Allow privilege escalation	10	0	13	76%
Medium	Allowed hostPath	2	0	12	12%
Medium	Automatic mapping of service account	55	0	55	100%
Medium	CVE-2022-0492-cgroups-container-escape	9	0	12	82%
Medium	Cluster internal networking	5	0	5	100%
Medium	Configured liveness probe	6	0	12	50%
Medium	CoreDNS poisoning	3	0	68	4%
Medium	Delete Kubernetes events	3	0	68	4%
Medium	Exec into container	1	0	68	1%
Medium	Forbidden Container Registries	4	0	12	38%
Medium	Host PID/IPC privileges	1	0	12	6%
Medium	HostNetwork access	2	0	12	12%
Medium	HostPath mount	3	0	12	18%
Medium	Images from allowed registry	9	0	12	75%
Medium	Ingress and Egress blocked	12	0	12	100%
Medium	Linux hardening	11	0	12	87%
Medium	Mount service principal	3	0	12	18%
Medium	Namespace without service accounts	4	0	48	8%
Medium	Network mapping	5	0	5	100%
Medium	No impersonation	1	0	68	1%
Medium	Non-root containers	11	0	12	94%
Medium	Portforwarding privileges	1	0	68	1%
	Configured readiness probe	7	0	12	63%
	Immutable container filesystem	10	0	12	81%
	K8s common labels usage	12	0	12	100%
	Label usage for resources	8	0	12	62%
	Naked PODs	2	0	20	10%
	Pods in default namespace	4	0	12	38%
	Resource policies	10	0	12	74%
	RESOURCE SUMMARY	186	0	163	26.76%

Secret not encrypted at rest

Use EncryptionConfiguration resource to encrypt Secret at rest

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
 - resources:
     - secrets
   providers:
     - aesgcm:
         keys:
           - name: key1
             secret: c2VjcmV0IGlzIHNlY3VyZQ==
     - identity: {}
```

Not setting requests and limits

Configuration Mistakes

Define resource requests and limits for all pods

- ensures proper resource allocation
- avoid resource contention

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: podinfo
spec:
  containers:
  - name: podinfo
    image: stefanprodan/podinfo
    resources:
      limits:
        cpu: 1m
        memory: 16Mi
      requests:
        cpu: 1m
        memory: 16Mi
```

Using the latest tag

Configuration Mistakes

- Use specific image tags and update them regularly
- Do not use latest tag as tomorrow's latest can be different than today's

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
  - name: www
    image: nginx:latest
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
  - name: www
    image: nginx:1.24
```

Using the default Namespace for all resources

Configuration Mistakes

Organize resources using Namespaces based on environment, team, or application to avoid conflicts and management issues

```
$ kubectl get ns
NAME                   STATUS   AGE
argocd                 Active   36d
cert-manager           Active   36d
default                Active   36d
events-exporter        Active   30d
kube-node-lease        Active   36d
kube-public            Active   36d
kube-system            Active   36d
local-path-provisioner Active   36d
myapp                  Active   35d
nats                   Active   36d
traefik                Active   36d
```

Applications are often deployed in their own namespaces

Not using livenessProbe and readinessProbe

Configuration Mistakes

Configure liveness and readiness probes to automatically manage pod health

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: podinfo
spec:
  containers:
  - name: podinfo
    image: stefanprodan/podinfo
    livenessProbe:
      httpGet:
        path: /healthz
        port: 9898
      initialDelaySeconds: 3
      periodSeconds: 10
    readinessProbe:
      httpGet:
        path: /readyz
        port: 9898
      initialDelaySeconds: 3
      periodSeconds: 5
```

- A livenessProbe triggers the restart of a container if it fails
- A readinessProbe makes sure a container is ready to receive traffic

Hardcoding configurations in Pods

Configuration Mistakes

Use ConfigMaps and Secrets to manage configurations externally

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  nginx.conf: |
    user www-data;
    worker_processes 4;
    pid /run/nginx.pid;
    events {
      worker_connections 768;
    }
    http {
      server {
        listen *:80;
        location / {
          proxy_pass http://api:5000;
        }
      }
    }
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: proxy
spec:
  containers:
  - name: proxy
    image: nginx:1.24
    ports:
    - containerPort: 80
    volumeMounts:
    - name: config
      mountPath: "/etc/nginx/"
  volumes:
  - name: config
    configMap:
      name: nginx-config
```

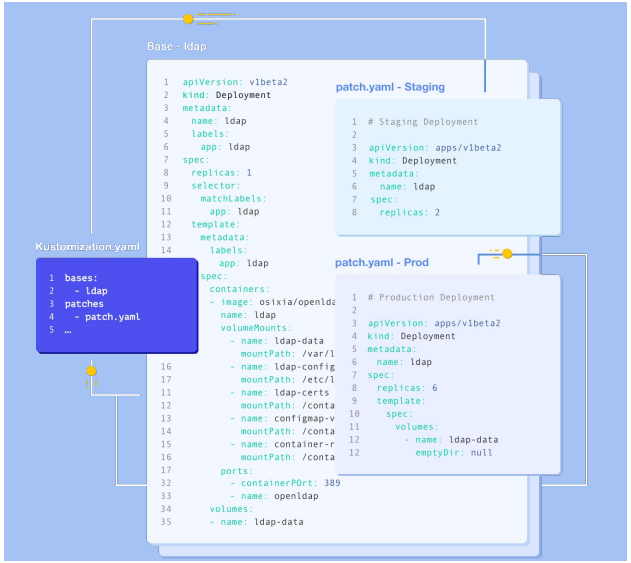
Inconsistent configuration across environments

Configuration Mistakes

Standardize configurations using tools like Helm or Kustomize

```
$ tree my-app/
my-app/
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

Using Helm to deploy an application



Using Kustomize to deploy an application

Not using automatic scaling

Configuration Mistakes

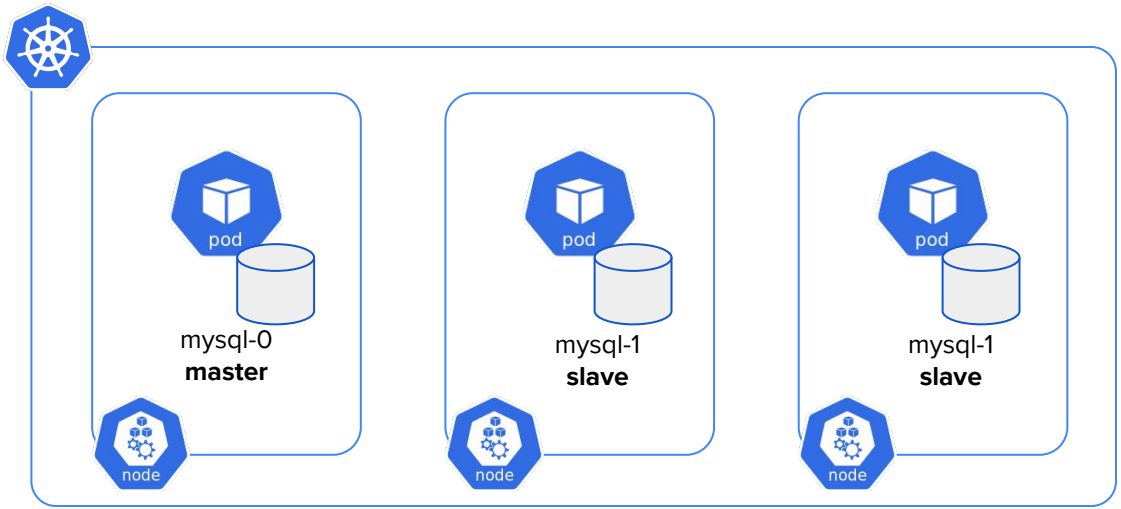
Configure Horizontal Pod Autoscaler (HPA) to scale applications based on metrics

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-v2
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: www
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

Note: Keda (CNCF Graduated project) https://keda.sh extends the functionalities of the HPA

Not using anti-affinity rules for Pods

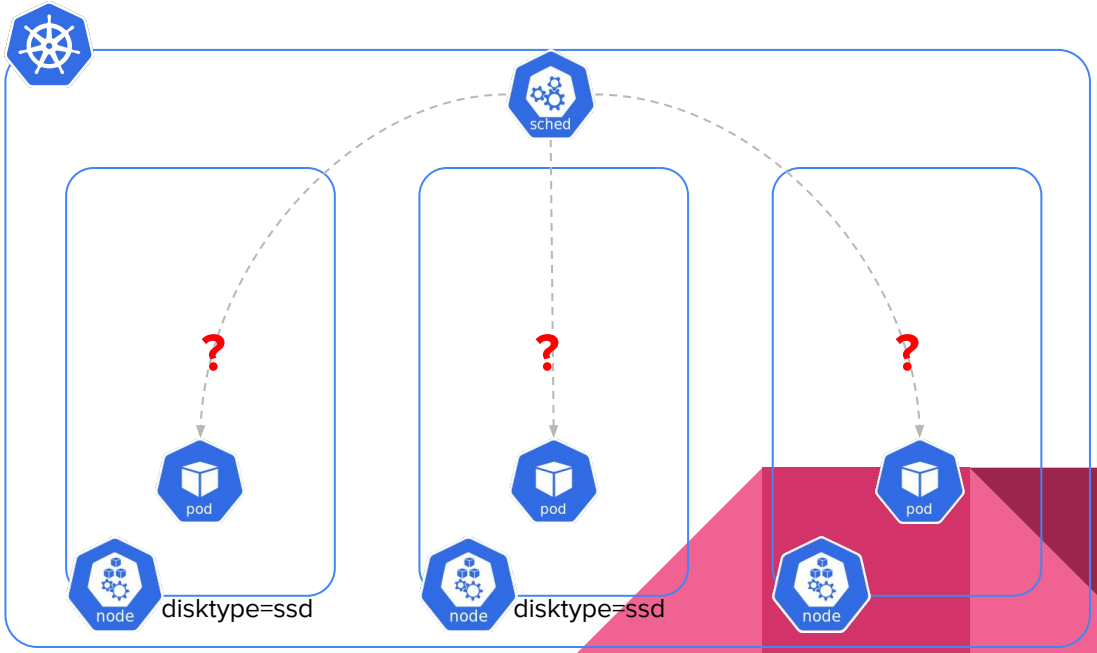
- Configure anti-affinity rules to distribute pods across nodes
- Prevents Pods from being placed on the same nodes which could lead to high availability issues



Deploying a Pod to the wrong Node

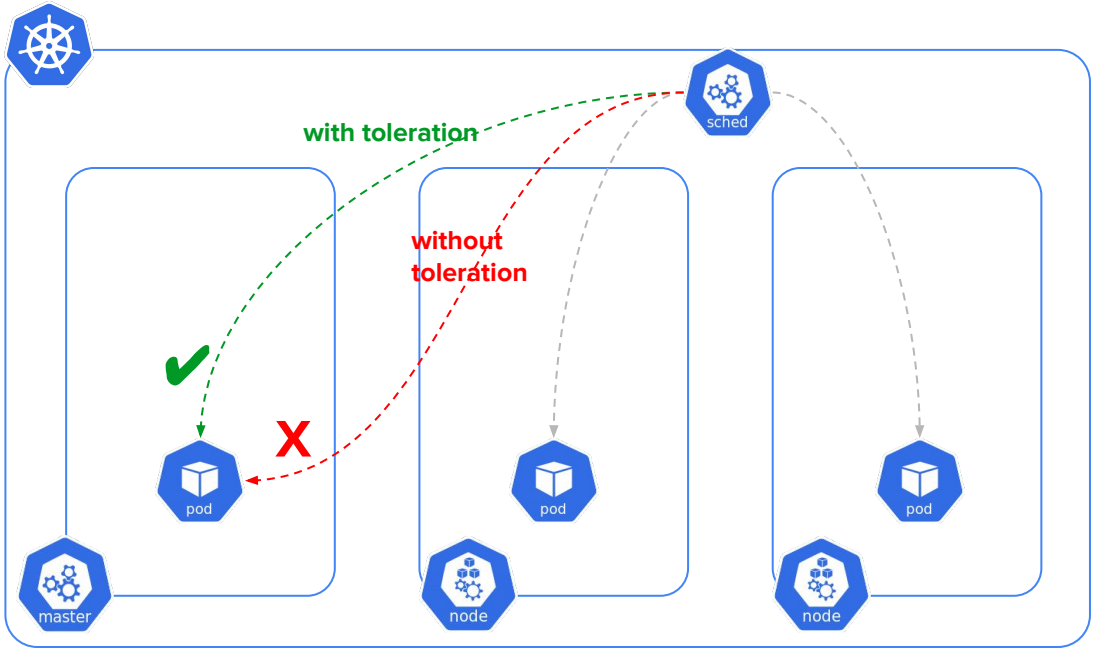
Use appropriate rules / properties for Pod placement

- nodeSelector
- nodeAffinity
- podAffinity / podAntiAffinity
- topologySpreadConstraints
- taint / toleration
- ressources disponibles
- priorityClass
- runtimeClass



Not using Taints and Tolerations correctly

Taints and tolerations control Pod placement based on node conditions



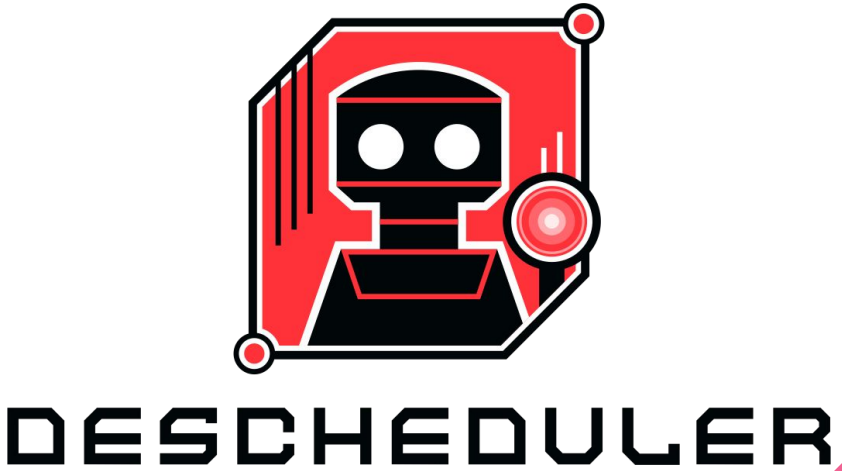
The controlplane node has the default taint node-role.kubernetes.io/controlplane:NoSchedule

Pods which do not tolerate the Taints cannot be scheduled on the controlplane

Overloading specific nodes

Use balanced pod placement strategies and monitor node utilization

Exemple: Descheduler https://github.com/kubernetes-sigs/descheduler



Ignoring pod priority and preemption

Set pod priorities and enable preemption to ensure critical pods are scheduled

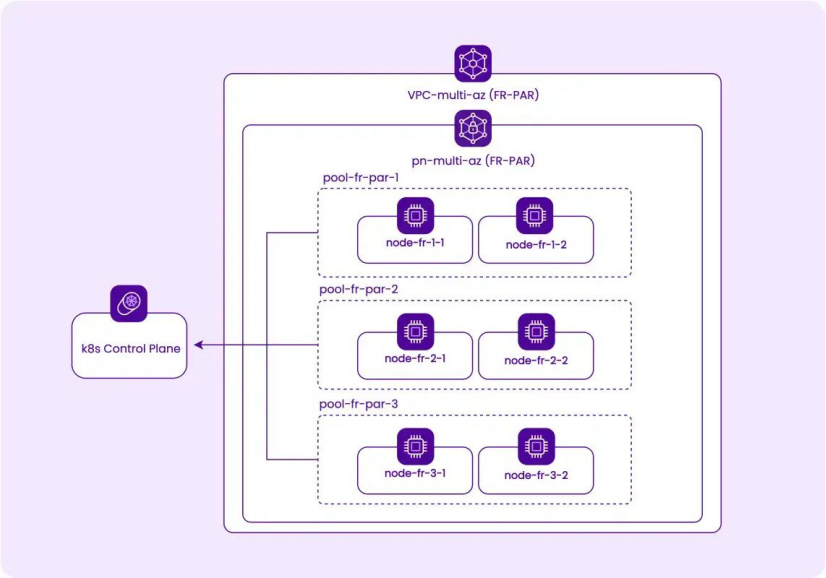
```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: critical-priority
value: 1000
globalDefault: false
description: "Class for critical pods"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: critical-pod
spec:
  priorityClassName: critical-priority
  containers:
  - name: critical-container
    image: tech/api:1.24
```

Note: preemption enabled by default, can be disabled on per Pod basis, eg to make sure a critical pod is never replaced

Not considering geographic placement of nodes

Deploy nodes across multiple clouds / regions / zones for high availability



Deployment of a cluster across several AZ in the same region (source Scaleway)

Lack of monitoring and logging

Implement a monitoring and logging solutions and use centralized logging systems

Lack of alerting mechanisms

Observability Mistakes

Set up alerting rules based on monitoring metrics to notify of potential issues

AlertManager

Not enabling Kubernetes auditing

Observability Mistakes

- Enable audit logs to keep track of requests done against the API Server
- Available stages
  - RequestReceived / ResponseStarted / ResponseComplete / Panic
- Available levels
  - None / Metadata / Request / RequestResponse

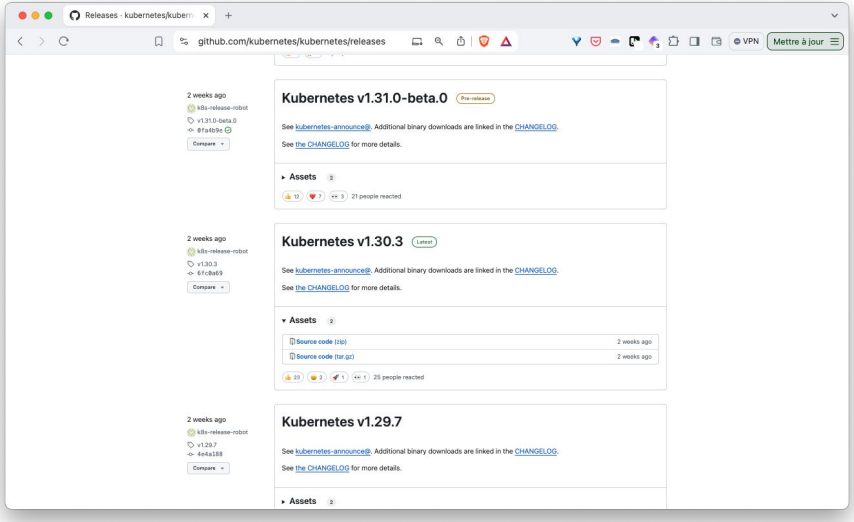
```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
```

Sample Policy logging all requests at the Metadata level

Not following the latest version and upgrading often

Cluster Management Mistakes

Regularly upgrade to the latest stable version of Kubernetes as running outdated Kubernetes versions can lead to security and compatibility issues



Using outdated APIs

Cluster Management Mistakes

Migrate to the latest stable APIs as they become available as using deprecated APIs can lead to future compatibility issues

```
$ kubectl api-versions
apps/v1
authentication.k8s.io/v1
authorization.k8s.io/v1
autoscaling/v1
autoscaling/v2
…
policy/v1
rbac.authorization.k8s.io/v1
scheduling.k8s.io/v1
storage.k8s.io/v1
v1
```

Not using managed services

Consider using managed Kubernetes services as Managing Kubernetes clusters manually increases operational overhead

Manual vs managed clusters

Not using IaC

Cluster Management Mistakes

- Use IaC tools like Terraform or Pulumi instead of manual cluster management (which can lead to inconsistencies and errors)
- IaC tools allow to keep cluster specification in VCS (Git)

```yaml
name: sks
runtime: yaml
description: SKS cluster management
outputs:
  kubeConfig: ${kubeconfig.kubeconfig}
resources:
  …
  cluster:
    type: exoscale:SksCluster
    properties:
      autoUpgrade: false
      cni: cilium
      description: A Kubernetes cluster on Exoscale
      exoscaleCcm: true
      exoscaleCsi: true
      metricsServer: true
      serviceLevel: starter
      name: sks-${pulumi.stack}
      zone: ${zone}
      version: ${version}
  nodepool:
    type: exoscale:SksNodepool
    properties:
      clusterId: ${cluster.id}
      name: sks-${pulumi.stack}-${nodepoolSuffix}
      zone: ${cluster.zone}
      instanceType: ${instanceType}
      size: ${size}
      securityGroupIds:
      - ${securityGroup.id}
```

Extract of Pulumi YAML descriptor

Not employing deployment models

Application Deployment Mistakes

- Use deployment models like blue-green, canary, or rolling updates deployment strategies to reduce downtime and risk
- Argo Rollout (https://argoproj.github.io/rollouts/) provides advanced upgrade strategies

Application Deployment Mistakes

Not scanning manifests before applying them

- Trivy (22k+ stars) - https://github.com/aquasecurity/trivy
- kubesec (1.2k stars) - https://github.com/controlplaneio/kubesec
- kube-score (2.7k stars) - https://github.com/zegl/kube-score
- checkov (6.8k stars) - https://github.com/bridgecrewio/checkov
- …

Not managing configurations separately from code

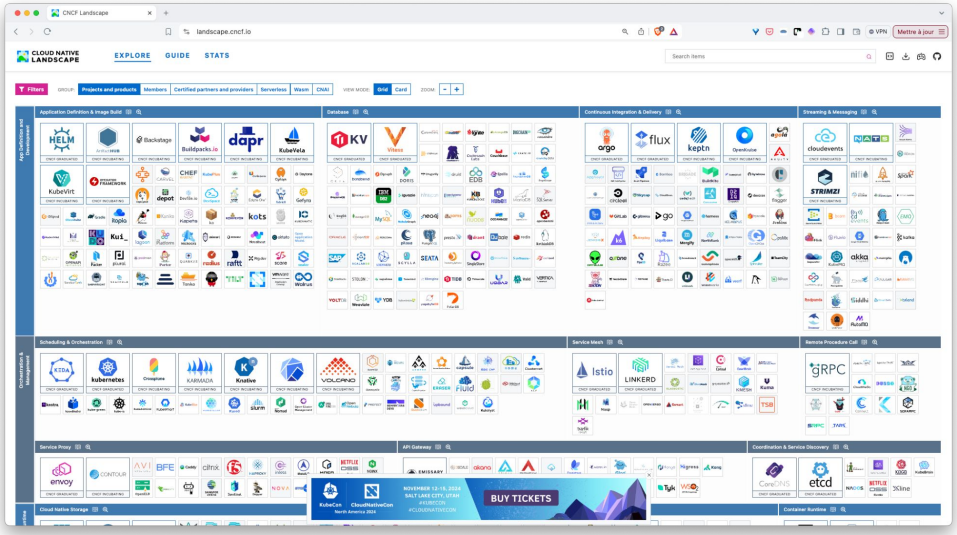
Application Deployment Mistakes

Use ConfigMaps and Secrets to manage configurations separately from application code as hardcoded configurations is bad practice in term of security plus it impacts portability

Not following CNCF ecosystem

CLOUD NATIVE COMPUTING FOUNDATION

CNCF hosts different categories of projects (storage / networking / observability / security / database / …)



CNCF landscape is huge, it contains many projects. Some of them could probably be integrated in your tech stack https://landscape.cncf.io

This is a non-exhaustive list
Suggestions are welcome