sysctl профиль NICE.OS CLOUD

NICE.OS CLOUD — преднастройка системы для Kubernetes и контейнерных нагрузок

Эта статья — практическое руководство для тех, кто хочет, чтобы узлы **NICE.OS CLOUD** работали стабильно, безопасно и быстро под контейнерными нагрузками. Мы разберём готовый набор `sysctl`, объясним зачем каждая опция нужна, дадим проверочные команды и покажем, что ещё можно подкрутить: от сети и дисков до `containerd`/`CRI-O`, `kubelet` и `systemd`.

✅ **Коротко:** ниже — готовый профиль «безопасность + производительность» для контейнер-хостов. Он совместим с `containerd`/`CRI-O`, kube-proxy, CNI (включая eBPF/Cilium), overlay/bridge-сетями, ingress/NodePort и типовым мониторингом.

1) Готовый `sysctl` для узла Kubernetes/контейнер-хоста 🚀

Сохраните файл как `/etc/sysctl.d/99-niceos-cloud.conf`, примените `sysctl --system`. После — проверьте ключевые параметры командами из следующего раздела.

```
# ============================================================
# NICE.OS CLOUD — финальные sysctl для узла Kubernetes/контейнер-хоста
# Безопасность + производительность контейнерных нагрузок
# Совместимо с containerd/CRI-O, kube-proxy, CNI (в т.ч. eBPF/Cilium),
# overlay/bridge-сетями, ingress/NodePort, мониторингом.
# ============================================================

# ----------------------------
# 🔒 Базовая безопасность ядра/памяти
# ----------------------------
kernel.randomize_va_space = 2
kernel.kptr_restrict = 2
```

```
kernel.dmesg_restrict = 1
kernel.sysrq = 0
fs.suid_dumpable = 0
vm.mmap_min_addr = 65536
vm.unprivileged_userfaultfd = 0
kernel.panic_on_oops = 1
kernel.panic = 10

# ----------------------------
# 🛡 ptrace / perf / eBPF — безопасно, но не мешает наблюдаемости
# ----------------------------
kernel.yama.ptrace_scope = 1
kernel.perf_event_paranoid = 2
kernel.unprivileged_bpf_disabled = 1
net.core.bpf_jit_harden = 2

# ----------------------------
# 📈 Масштабируемость PID/дескрипторов/наблюдения
# ----------------------------
kernel.pid_max = 262144
fs.file-max = 2097152
fs.inotify.max_user_instances = 8192
fs.inotify.max_user_watches = 1048576
vm.max_map_count = 262144

# ----------------------------
# 🌐 Сетевой стек — безопасность + overlay/bridge/eBPF
# ----------------------------
net.ipv4.ip_forward = 1
net.ipv6.conf.all.forwarding = 1

net.ipv4.conf.all.rp_filter = 2
net.ipv4.conf.default.rp_filter = 2

net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0

net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.default.log_martians = 1
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1

net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 8192
```

```
net.ipv4.tcp_rfc1337 = 1
net.ipv4.tcp_challenge_ack_limit = 1073741823

net.ipv4.neigh.default.gc_thresh1 = 2048
net.ipv4.neigh.default.gc_thresh2 = 4096
net.ipv4.neigh.default.gc_thresh3 = 8192
net.ipv6.neigh.default.gc_thresh1 = 2048
net.ipv6.neigh.default.gc_thresh2 = 4096
net.ipv6.neigh.default.gc_thresh3 = 8192

net.core.somaxconn = 32768
net.core.netdev_max_backlog = 16384
net.ipv4.ip_local_port_range = 1024 65000

# ----------------------------
# 🔗 nf_conntrack — ключевой ресурс для сервисного трафика
# ----------------------------
net.netfilter.nf_conntrack_max = 1048576
# net.netfilter.nf_conntrack_tcp_timeout_established = 432000
# net.netfilter.nf_conntrack_tcp_timeout_fin_wait = 60
# net.netfilter.nf_conntrack_tcp_timeout_close_wait = 60
# net.netfilter.nf_conntrack_tcp_timeout_time_wait = 120

# ----------------------------
#  Bridge netfilter — правила iptables/nft на бридже
# ----------------------------
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1

# ----------------------------
# 🌍 IPv6 — предсказуемое поведение на роутирующем узле
# ----------------------------
net.ipv6.conf.all.accept_ra = 0
net.ipv6.conf.default.accept_ra = 0

# ----------------------------
# ⚙️ Опционально: тюнинг стека для производительности
# ----------------------------
# net.core.default_qdisc = fq
# net.ipv4.tcp_congestion_control = bbr
# net.ipv4.ip_nonlocal_bind = 1   # если требуется kube-proxy/ipvs
```

Как применить ⚙️

```
sudo sysctl --system
```

Проверьте выборочно:

```
sysctl net.ipv4.ip_forward
sysctl net.bridge.bridge-nf-call-iptables
sysctl net.netfilter.nf_conntrack_max
sysctl fs.inotify.max_user_watches
```

Параметры, начинающиеся на `net.bridge.*`, требуют загруженного модуля `br_netfilter` (см. «Загрузка модулей» ниже).

Что это даёт ▦

- Более предсказуемое и безопасное поведение ядра.
- Готовность сети к overlay/bridge, ingress/NodePort и eBPF.
- Запас по PID/FD/inotify под десятки/сотни подов.
- Высокие пороги `conntrack` для интенсивного север/юг трафика.

2) Разбор настроек простыми словами

Ниже — ключевые группы параметров и зачем они нужны. Важно: значения из профиля подобраны как «безопасный дефолт для прод-узла среднего масштаба». Настраивайте дальше по телеметрии.

🔒 Безопасность ядра и памяти

- `ASLR=2`, скрытие адресов ядра и `dmesg` — меньше пищи для эксплойтов.
- Запрет `userfaultfd` для не-root — снижает риск race-атак.
- `panic_on_oops` + авто-перезагрузка — узел быстро «вылечится», кластер перераспределит нагрузку.

Когда менять?

Обычно не нужно. Для отладки можно временно ослаблять ограничения на тестовых узлах.

🛡 ptrace / perf / eBPF

- `ptrace_scope=1` — разработчики могут дебажить процесс своего UID (удобно в подах), но не больше.
- `perf_event_paranoid=2` — наблюдаемость без сырых выборок ядра для не-root.

- `unprivileged_bpf_disabled=1` + `bpf_jit_harden=2` — eBPF только для привилегированных агентов (Cilium, Falco и т.п.).

Когда менять?

Если нужны не-привилегированные eBPF-проги (редко) — осознанно ослабляйте на dev.

🧱 PID/FD/inotify/VM

- `pid_max=262144` — запас по процессам для множества подов и сайдкаров.
- `file-max=2M` — много соединений/файлов — не проблема.
- `inotify` увеличен — довольны логгеры, агенты, сборщики метрик.
- `vm.max_map_count=262144` — для JVM/Elastic/ClickHouse и интенсивных mmap.

Когда менять?

Если упираетесь — увеличивайте. Следите за ulimit процесса (см. «limits.conf» ниже).

🌐 Сеть, SYN-защита и backlog'и

- Форвардинг IPv4/IPv6 — хост умеет маршрутизировать трафик подов/overlay.
- `rp_filter=2` — «loose» режим для асимметрии/overlay/VRF, типично для кластеров.
- Запрет redirect/source-route, лог «марсиан» — гигиена безопасности.
- `somaxconn` и `netdev_max_backlog` повышены — меньше дропов под пиками.
- Эфемерные порты расширены — много NAT-соединений (NodePort/egress) не упрутся в лимит.

Когда менять?

Под пиковые ingress/egress нагрузки — увеличить ещё. Опираться на метрики дропов/квев.

🔗 conntrack

- `nf_conntrack_max=1048576` — ~1M записей: безопасный старт для средних прод-узлов.
- Таймауты TCP можно уменьшать осторожно, если записи «заканчиваются».

Когда менять?

Смотрите метрики: заполнение таблицы, дропы, rehash. Увеличивайте до исчезновения дропов.

Bridge netfilter и IPv6

- `bridge-nf-call-iptables=1` — правила на бридже для Kubernetes (iptables/nft).
- IPv6: явный `accept_ra=0` при `forwarding=1` делает поведение предсказуемым.

Важно

Не забудьте загрузить модуль `br_netfilter` (см. ниже).

3) Что ещё стоит настроить на контейнер-хосте

Эти рекомендации дополняют `sysctl` и дают устойчивый прирост в реальных кластерах.

🎁 Контейнерный runtime и cgroups

- **cgroup v2** как единая иерархия — современная изоляция и честные квоты CPU/IO.
- **containerd/CRI-O**: включите *systemd cgroup driver* (совместимость с kubelet).
- Выдавайте real-лимиты (CPU/Memory/IO) на уровне Pod/Container — меньше «noisy neighbor».

```
# containerd (пример)
# /etc/containerd/config.toml
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
  runtime_type = "io.containerd.runc.v2"
  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
    SystemdCgroup = true
```

Проверьте: `kubectl describe node <node>` — в разделе *Non-terminated Pods* видны лимиты/requests.

swap/THP/overcommit (под загрузки общего профиля)

- **Swap** для Kubernetes обычно выключают (если не включали swap-саппорт явным решением).

- **Transparent HugePages** — часто рекомендуют *madvise* или *never* для низкой латентности.
- **vm.overcommit_memory** — по умолчанию нормально; для БД и in-memory систем — тюнинг аккуратно.

```
# Отключить swap (если политика кластера требует):
sudo swapoff -a
# и убрать из /etc/fstab

# THP (временно, до ребута):
echo madvise | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
```

Для специализированных нагрузок (БД/аналитика) следуйте вендорным гайдам.

IRQ, сеть и очереди

- **irqbalance** включён — распределяет прерывания по CPU.
- Настройте RPS/XPS/RFS для высокоскоростных интерфейсов; увеличьте `rx/tx ring` (через `ethtool`).
- Включите **BBR** + `fq` (если политика допускает) — меньшая латентность под пиками.

```
# Пример (опционально, если разрешено политикой):
echo fq | sudo tee /proc/sys/net/core/default_qdisc
echo bbr | sudo tee /proc/sys/net/ipv4/tcp_congestion_control

# Примеры с ethtool (подставьте интерфейс):
sudo ethtool -g eth0          # посмотреть ring
sudo ethtool -G eth0 rx 4096 tx 4096
sudo ethtool -k eth0          # offload'ы
```

📚 systemd-limits и journald

- Поднимите лимиты дескрипторов/процессов для сервисов (containerd, kubelet, прокси и т.п.).
- Ограничьте объём журнала и задайте ротацию, чтобы не «съесть» диск.

```
# /etc/security/limits.d/90-niceos.conf
* soft nofile 1048576
* hard nofile 1048576
* soft nproc  262144
* hard nproc  262144
```

```
# /etc/systemd/journald.conf (фрагмент)
SystemMaxUse=2G
RuntimeMaxUse=1G
MaxFileSec=1day
```

Перезапустите `systemd-journald` после изменения конфигурации.

Загрузка модулей ядра (Kubernetes-must)

- `br_netfilter` — нужен для правил на бридже (iptables/nft) в K8s.
- `overlay` — драйвер overlayfs для контейнеров.

```
# /etc/modules-load.d/k8s.conf
br_netfilter
overlay
```

После — проверьте `lsmod | egrep 'br_netfilter|overlay'`.

🔐 SELinux/AppArmor и firewall

- Рекомендуется **enforcing** (или AppArmor профили) — меньше blast-radius.
- Сетевой доступ — открывайте только нужное (API-сервер, kubelet, etcd, ingress, nodeport).

Политики безопасности — часть SRE-культуры. Автоматизируйте с Ansible/Helm/OPA/Gatekeeper.

4) Проверка и наблюдаемость 📊

После применения настроек — важно убедиться, что узел действительно стал устойчивее.

🔎 Быстрые проверки

```
# Маршрутизация и bridge:
sysctl net.ipv4.ip_forward
sysctl net.bridge.bridge-nf-call-iptables

# Пулы PID/FD/inotify:
sysctl kernel.pid_max fs.file-max fs.inotify.max_user_watches
```

```
# Conntrack:
sysctl net.netfilter.nf_conntrack_max
cat /proc/sys/net/netfilter/nf_conntrack_count  # текущее заполнение
```

Смотрите в мониторинге: дропы пакетов, дропы в backlog, заполнение conntrack, latency пиков.

Нагрузочные мини-тесты

- Ingress/NodePort: сгенерируйте пиковые SYN — проверьте `tcp_max_syn_backlog`, дропы и latency.
- Много мелких файлов/логов: проверьте число дескрипторов и инстансов inotify.
- Много соединений наружу: посмотрите диапазон эфемерных портов, conntrack и egress-NAT.

5) Чек-лист внедрения ✔

- Создать `/etc/sysctl.d/99-niceos-cloud.conf` и применить `sysctl --system`.
- Загрузить модули `br_netfilter` и `overlay` через `/etc/modules-load.d/k8s.conf`.
- Включить *SystemdCgroup=true* для `containerd`/`CRI-O`.
- Отрегулировать `limits.conf` и `journald.conf` (FD/CPU/журналы).
- Проверить swap/THP согласно политике кластера.
- Поставить и настроить мониторинг сети, conntrack, backlog, latency.

- Проверить kubelet и CNI (bridge/overlay/eBPF) на тестовом пуле узлов.
- Смоделировать пик трафика: ingress/NodePort/egress, убедиться в отсутствии дропов.
- Собрать телеметрию за неделю и при необходимости увеличить `conntrack_max`, backlog’и.
- Зафиксировать профиль как «золотой образ» узла в CI/CD (Immutable узлы).

📎 **Совет:** храните профиль узла как код (Git), применяйте через Ansible/Helm/Kustomize. Любые изменения — через pull-request и автотесты.

6) Вопросы и ответы 💬

Нужно ли включать BBR и `fq` всем? 🛠️

Это опциональные тюнинги для снижения латентности и улучшения поведения под пиками. Часто дают пользу, но согласуйте с политикой компании и протестируйте под вашу нагрузку. Для строго регламентированных сред оставьте значения по умолчанию из профиля.

Что, если `conntrack` всё равно забивается? 🔪

Увеличьте `nf_conntrack_max`, при необходимости аккуратно уменьшайте таймауты устойчивых состояний, оптимизируйте таймауты приложений, балансировку и паттерны соединений. Следите за метриками *conntrack count/limit/drops*.

Swap выключать обязательно?

Для Kubernetes — да, если вы не включили явный *swap support* и не измерили его пользу. В контейнерных кластерах предсказуемость лучше достигается без swap.

Как катить это на весь кластер без ручной рутины? ♻️

Держите профиль узла в репозитории, применяйте конфигурацию через Ansible/TF/Flux/ArgoCD. Пул новых узлов должен подниматься уже с нужным профилем (immutable образ). Никаких ручных шагов.

Итог 🏁

Профиль **NICE.OS CLOUD** даёт безопасные и быстрые дефолты для контейнер-хостов: надёжный сетевой стек для overlay/bridge и ingress/NodePort, устойчивость под пиками, приличный запас по PID/FD/inotify, а также базовую гигиену безопасности ядра и eBPF.

Дальше — дело телеметрии: наблюдайте, где именно возникают узкие места, и масштабируйте значения точечно. Храните настройки как код, катите через CI/CD и держите узлы неизменяемыми. Так контейнерная платформа остаётся предсказуемой, а команда — спокойной. 🖤