ГАЙД ДЛЯ НОВИЧКОВ · 20 СЛАЙДОВ · С НУЛЯ

Ты запустил его. Но не защитил.

KUBERNETES ОСНОВЫ БЕЗОПАСНОСТИ

По умолчанию он открыт.
И об этом редко предупреждают новичков.

CLUSTER | SECURITY | POSTURE

Network isolation — OFF
Ограничение сетевого доступа между pod и сервисами.

RBAC least privilege — OFF
Минимально необходимые права доступа.

Non-root containers — OFF
Запуск контейнеров не от root.

Secret encryption at rest — OFF
Шифрование secret в хранилище.

Audit logging — OFF
Аудит действий в кластере.

Admission policy — OFF
Политики контроля и валидации.

6 из 6 защитных механизмов отключены по умолчанию.

```
kubectl get nodes
NAME     STATUS   ROLES
node-1   Ready    <none>
node-2   Ready    <none>
node-3   Ready    <none>

apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: app
  image: ngin
```

ЛИСТАЙТЕ → | от простого развертывания к безопасному кластеру

ЧТО ТАКОЕ KUBERNETES

Kubernetes — это автопилот для ваших контейнеров.
Вы говорите, что хотите запустить, а он сам решает, где разместить, поддерживает их в рабочем состоянии и масштабирует по необходимости.



БЕЗ KUBERNETES
вы управляете всем вручную



размещаете контейнеры вручную и перезапускаете их, когда они падают

С KUBERNETES
он делает эту работу за вас



планирует размещение, следит за состоянием и масштабирует по нагрузке

SCHEDULES
размещает контейнеры на подходящих узлах

HEALS
автоматически перезапускает упавшие контейнеры

SCALES
добавляет копии при увеличении нагрузки

ОСНОВНЫЕ БЛОКИ KUBERNETES

Четыре ключевых компонента, которые объясняют всю систему. Каждый находится внутри следующего, как вложенные контейнеры.

CLUSTER
Весь кластер, единая система управления.

Набор узлов (nodes), который работает как один кластер и управляется control plane.

NODE
Узел кластера.

Физическая или виртуальная машина (VM), на которой запускаются pod'ы.

POD
Группа контейнеров.

Один или несколько контейнеров, которые делят сеть (IP) и хранилище и запускаются вместе.

CONTAINER
Контейнер с приложением.

Упакованное приложение со всеми зависимостями, которое запускается внутри pod'а.

ИЗ ЧЕГО СОСТОИТ KUBERNETES

Кластер Kubernetes состоит из двух частей: мозга, который принимает решения, и рабочих узлов, которые запускают ваши приложения.



CONTROL PLANE мозг кластера

API server
входная точка для всех запросов

etcd
хранит всё состояние кластера

scheduler
выбирает узел (node) для каждого pod

controller mgr
поддерживает нужное состояние кластера

NODE 1 рабочий узел

kubelet
запускает и следит за pod на узле

kube-proxy
настраивает сетевые правила

runtime
запускает контейнеры (например, containerd)

NODE 2 рабочий узел

kubelet
запускает и следит за pod на узле

kube-proxy
настраивает сетевые правила

runtime
запускает контейнеры (например, containerd)

NODE 3 рабочий узел

kubelet
запускает и следит за pod на узле

kube-proxy
настраивает сетевые правила

runtime
запускает контейнеры (например, containerd)

ГАЙД ДЛЯ НОВИЧКОВ · 20 СЛАЙДОВ · С НУЛЯ

КАК ВСЁ РАБОТАЕТ В KUBERNETES

Развёртывание приложения — это цепочка взаимодействий. Один запрос с вашего терминала запускает работающий контейнер.



01

kubectl apply

Вы отправляете желаемое состояние кластера (например, Deployment).

02

API server

Аутентифицирует запрос, проверяет его и сохраняет в кластер.

03

etcd

Хранит всё состояние кластера (желаемое и текущее).

04

scheduler

Выбирает подходящий узел (node) для запуска вашего pod.

05

kubelet

На выбранном узле получает задачу и запускает контейнер.

06

Pod running

Ваш контейнер запущен и приложение работает.

ПОЧЕМУ KUBERNETES НЕБЕЗОПАСЕН ПО УМОЛЧАНИЮ

Kubernetes настроен так, чтобы просто работать, а не быть защищённым. Свежий кластер отдаёт злоумышленнику слишком много возможностей.

Плоская сеть
Любой pod может достучаться до любого другого pod — без сегментации и firewall по умолчанию.

Избыточные права
Стандартные service account часто имеют больше прав, чем нужно.

Контейнеры от root
По умолчанию контейнеры запускаются от root, если явно не указано иначе.

Секреты в открытом виде
Данные часто хранятся в базе в base64, а не в зашифрованном виде.

Нет проверки при размещении
Любой манифест, который проходит парсинг, будет применён — без политики контроля (policy gate).

Открытый выход в интернет
По умолчанию pod может обращаться в любую точку сети — без ограничений egress.

Четыре двери для атак

Атакующие не используют магию. Они находят точки входа — и проходят через одну из четырёх дверей. Давайте разберём, какие именно.



Открытые API-сервисы

Публичные точки доступа с слабой аутентификацией или недостаточной защитой.

→ прямой доступ к внутренней логике

```
POST /api/v1/internal
{
  "token": "...",
  "action": "..."
}
200 OK
```

Открытые порты и панель

Сервисы, доступные извне, или публичные дашборды с лишними правами.

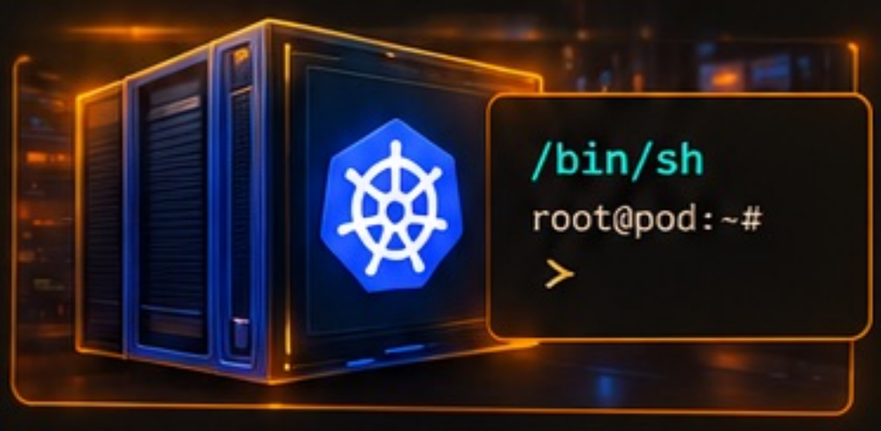
→ запуск команд внутри ваших pod-ов

```
$ nmap -p 22,80,443 ...
22/tcp   open   ssh
80/tcp   open   http
443/tcp  open   https
OPEN PORTS
```

Уязвимое приложение в pod-ах

Баги в коде или конфигурации контейнеров (например, RCE).

→ доступ к выполнению кода внутри pod-а



Заражённые образы

Бэкенд-образы или контейнеры, в которых уже есть вредоносный код.

→ запуск вредоносной логики при доступе



ЭФФЕКТ BLAST RADIUS

Вопрос не в том, «безопасен ли один pod».
Вопрос в том, «куда он может добраться,
если будет скомпрометирован».

5 Весь кластер под контролем

Атакующий двигается дальше и получает доступ к control plane из одного из узлов.

2 Кража токенов и секретов

Злоумышленник получает доступ к `service account`, секретам и другим чувствительным данным.



1 Компрометация pod

Уязвимость в приложении даёт злоумышленнику доступ к pod и возможность выполнить код (например, получить shell).

3 Доступ к другим pod

При плоской сети злоумышленник может перемещаться в другие pod без дополнительных ограничений.

4 Побег на узел (node)

Если pod обладает повышенными правами (например, `privileged`), атакующий может выйти на node и получить доступ к хосту.

КОГДА RBAC ДАЁТ СЛИШКОМ МНОГО ПРАВ

RBAC определяет, кто что может делать в кластере. Одна из самых частых ошибок — дать приложению гораздо больше прав, чем ему действительно нужно.



rolebinding.yaml — YAML

```yaml
kind:  ClusterRoleBinding
subjects:
  - kind: ServiceAccount
    name: my-app     # обычное приложение
roleRef:
   name: cluster-admin   # полный контроль (!)
```

ЧТО ЭТО ЗНАЧИТ

Любой, кто получит контроль над этим приложением, автоматически получает доступ ко всему кластеру. Один баг в приложении — и это полный захват кластера.



КАК ИСПРАВИТЬ

Назначайте минимально необходимые права (минимальный `role`), давайте только нужные **`verbs`** на конкретные ресурсы и в одном **`namespace`**. Никогда не используйте **`cluster-admin`** для приложений.



ПРИВИЛЕГИРОВАННЫЕ КОНТЕЙНЕРЫ

Привилегированный контейнер почти не изолирован от хоста. Если злоумышленник попадает в такой контейнер — вся нода оказывается под его контролем.



pod-escape.yaml YAML

```yaml
securityContext:
  privileged: true   # контейнер = хост (почти без изоляции)

$ mount /host    # получаем доступ к диску ноды
$ chroot /host bash   # выходим в файловую систему хоста
$ whoami
root   # уже на ноде, а не в pod
```

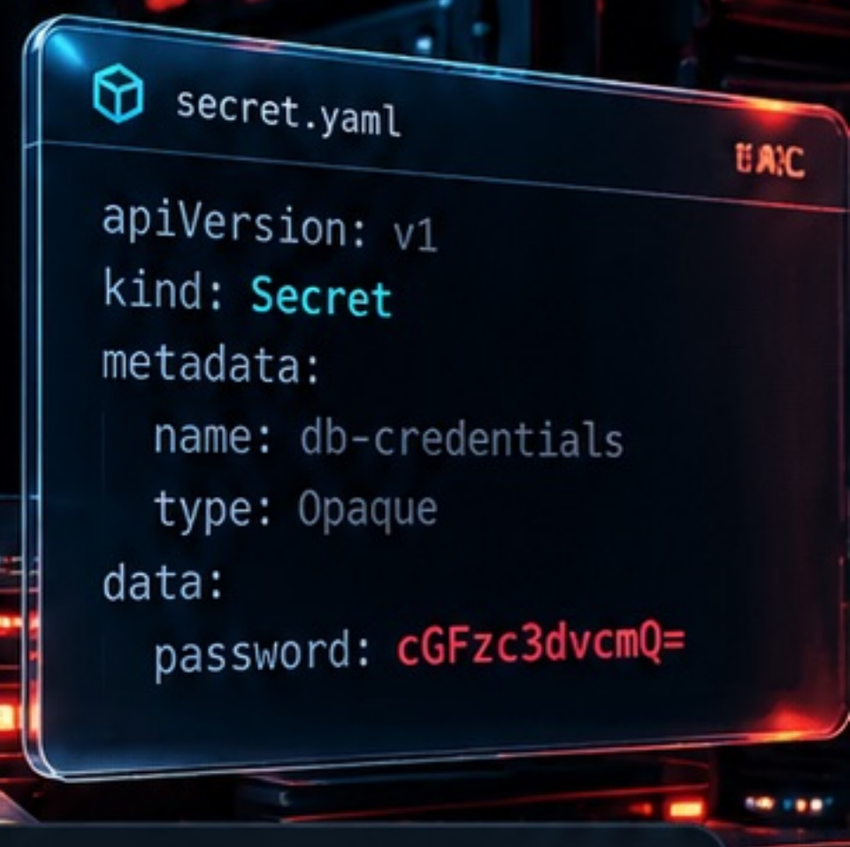
Привилегированный pod → Файловая система хоста (node) → root на ноде

КАК ИСПРАВИТЬ

Никогда не используйте **privileged**. Включайте **runAsNonRoot**, убирайте все лишние **Linux capabilities** и блокируйте повышение привилегий (**privilege escalation**).

СЕКРЕТЫ НЕ ЗАЩИЩЕНЫ

Kubernetes Secrets выглядят как будто защищены, но на самом деле только base64-кодированы. Это не шифрование.



secret.yaml — YAML

```yaml
kind: Secret
data:
  password: cGFzc3dvcmQ=    # base64

$ echo cGFzc3dvcmQ= | base64 -d
password    # декодируется одной командой
```

В etcd
Хранятся без шифрования по умолчанию.

В переменных окружения
Могут попасть в pod и быть выведены в логи.

Через RBAC
Часто доступны слишком широкому кругу ролей.

КАК ИСПРАВИТЬ

Шифруйте **secrets** «в покое», используйте отдельный **secret manager** (например, Vault, AWS Secrets Manager) и применяйте короткоживущие **tokens** вместо статичных значений.



ПЛОСКАЯ СЕТЬ

По умолчанию в кластере нет внутренних ограничений. Каждый pod может установить соединение с любым другим pod.

По умолчанию любой → любой

Каждый pod может достучаться до любого сервиса в кластере.



ЧЕМ ЭТО ОПАСНО

Если злоумышленник скомпрометирует один pod, он сможет сканировать и подключаться ко всем сервисам в кластере. Боковое перемещение **(lateral movement)** становится очень простым.

```
> kubectl exec -it pod
→ api.default
→ db.prod
→ redis.cache
→ internal.svc
→ kube-system
```

ОТКРЫТЫЙ CONTROL PLANE

Control plane работает всего на нескольких сетевых портах. Если любой из них доступен из интернета, кластер можно полностью скомпрометировать.



API server

Главная точка входа для управления всем кластером.

ПОРТ 6443

> атакующий отправляет команды с правами администратора

etcd

Сырая база данных всего кластера: все объекты, секреты и конфигурация.

ПОРТ 2379

> атакующий читает весь кластер, эксплоиты не нужны

kubelet

Агент на каждом узле, который запускает pods.

ПОРТ 10250

> атакующий выполняет код в любом pod на узле

КАК ИСПРАВИТЬ

Держите control plane в **приватной сети**. Закройте все порты на **firewall** и требуйте аутентификацию для каждого из них, особенно **etcd**.



ПРИНЦИП МИНИМАЛЬНЫХ ПРИВИЛЕГИЙ

Не давайте больше прав, чем нужно. В Kubernetes это означает: каждой идентичности — только минимальный набор разрешений.



role.yaml — YAML

```yaml
kind: Role          # namespaced, not cluster
rules:
  - resources: ["pods"]
    verbs: ["get", "list"]  # read only
```

ServiceAccount — идентичность → **RoleBinding** — назначает роль → **Role** — узкая, в одном namespace

Минимальные права — только необходимые разрешения

Namespaced roles — без cluster-wide привилегий

Без wildcard verbs — избегайте "*" в verbs

Аудит доступа — отслеживайте, кто и что может

ЗАКАЛЯЙТЕ POD

Несколько простых настроек могут превратить открытый контейнер в изолированный и значительно снизить ущерб при компрометации приложения.



secure-pod.yaml — YAML

```yaml
securityContext:
  runAsNonRoot: true        # запускаем не от root
  privileged: false         # без привилегированного режима
  readOnlyRootFilesystem: true   # файловая система только для чтения
  allowPrivilegeEscalation: false # запрет на повышение привилегий
  capabilities:
    drop: ["ALL"]           # убираем все лишние Linux capabilities
```

Запускайте от непривилегированного пользователя
Даже если приложение скомпрометировано, оно не будет иметь root-прав.

```
# whoami
appuser
```

Используйте read-only filesystem
Злоумышленник не сможет записывать инструменты или изменять содержимое контейнера.

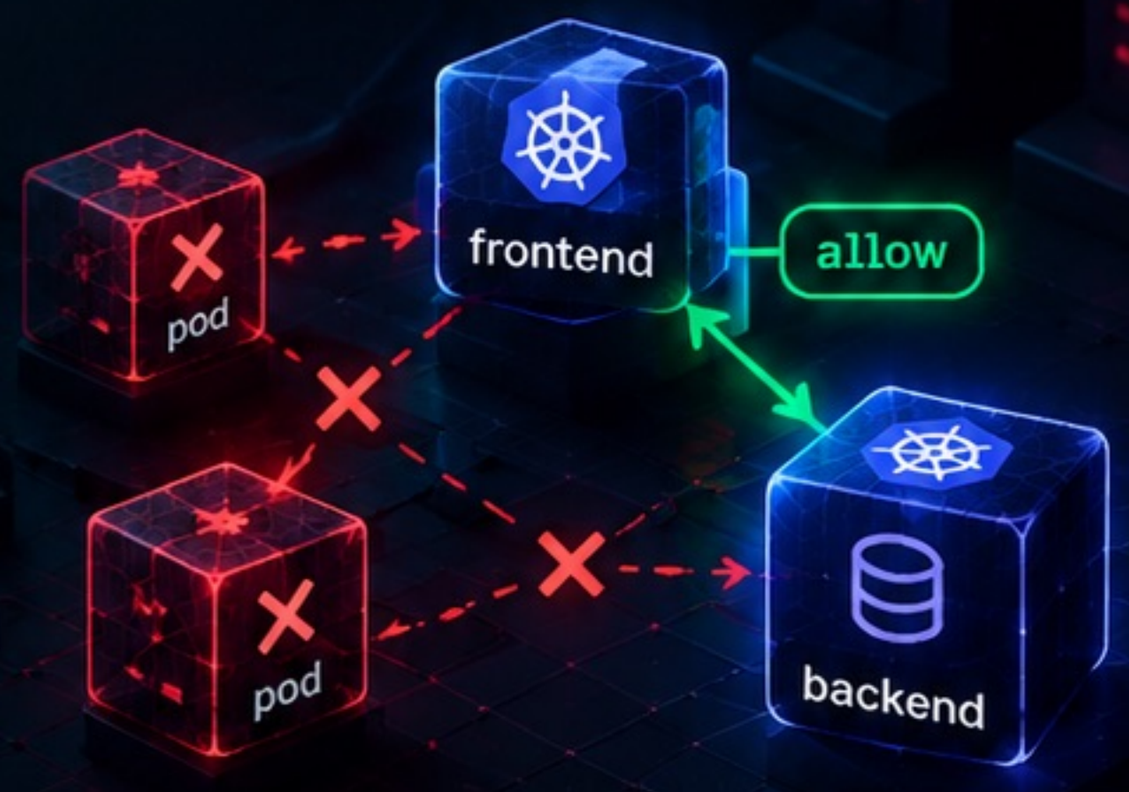


Запретите эскалацию привилегий и уберите лишние caps
Закрывает распространённые пути повышения привилегий к узлу (node) и другим сервисам.

```
CAP_SYS_ADMIN
CAP_NET_ADMIN
CAP_SYS_PTRACE
```

СЕТЕВЫЕ ПОЛИТИКИ

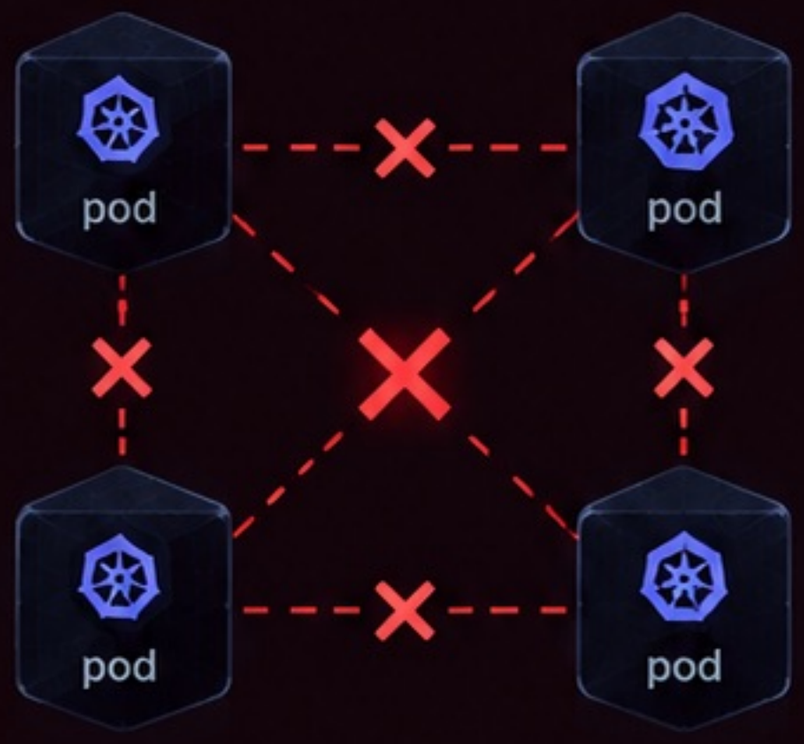
Исправление из слайда 12. Сначала блокируем весь трафик между pod, а затем явно разрешаем только нужные соединения.



ШАГ 1 default-deny

Блокируем всё

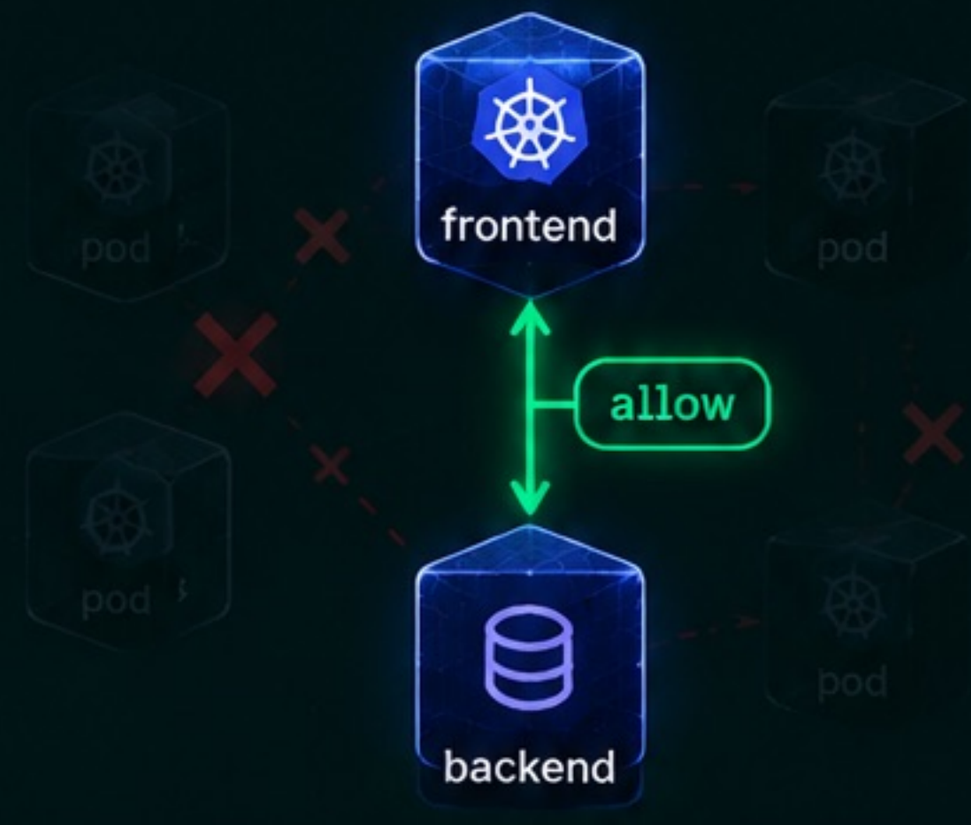
Запрещаем весь трафик между pod по умолчанию.



ШАГ 2 explicit allow

Разрешаем только нужное

Открываем ровно те соединения, которые действительно необходимы.



КАК ЭТО РАБОТАЕТ

Думайте о **NetworkPolicy** как о межсетевом экране между pod: по умолчанию **deny**, разрешаем только то, что действительно нужно (**by exception**).

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
   - Ingress
```

SECRETS И ОБРАЗЫ

Исправления из слайдов 11 и 7. Защищайте то, что хранят pod'ы, и доверяйте только тем образам, которые вы запускаете.



SECRETS

Храните чувствительные данные правильно.

Шифруйте в покое
Включите `encryption` для `Kubernetes Secrets` (например, через KMS).

Используйте отдельный secret manager
Например, `HashiCorp Vault`, `AWS Secrets Manager` и т.п.

Короткие TTL и ротация
Используйте `short-lived`, `rotated tokens`.

Не храните в env vars или git
Исключите попадание секретов в код и CI/CD.

IMAGES

Запускайте только проверенные образы.

Проверяйте образы на уязвимости
Сканируйте образы в `registry` (Trivy, Grype, Docker Scout и др.).

Подписывайте и проверяйте образы
Используйте подписи (Sigstore / Cosign) и policy (OPA).

Используйте фиксированные digest'ы
Не полагайтесь на `:latest`.

Минимальные base images
Используйте distroless, slim или официальные образы.





4 УРОВНЯ CLOUD NATIVE

Безопасность строится слоями. Каждый уровень закрывает следующий, а внутренние слои настолько же защищены, насколько надёжны внешние.



CLOUD
Инфраструктура, на которой работает ваш кластер.

IAM · VPC · Logging · Monitoring

CLUSTER
Управление доступом и сетевые политики.

RBAC · NetworkPolicy · Admission

CONTAINER
Запуск без root, минимальный набор пакетов, сканирование.

Non-root · Image scanning · Minimal

CODE
Ваше приложение: валидация входных данных, никаких секретов.

Input validation · No secrets · Least privilege

ЧЕКЛИСТ ПО УСИЛЕНИЮ

Две роли, два набора действий. Закройте оба — и ваш кластер перестанет быть открытой дверью.

КАК РАЗРАБОТЧИК
делаем контейнеры безопаснее

- **Запускайте контейнеры с runAsNonRoot**
  не запускайте от root
- **Убирайте лишние Linux capabilities**
  давайте только нужные
- **Не храните секреты в коде и env**
  используйте Secret manager
- **Сканируйте и подписывайте образы (images)**
  блокируйте использование уязвимых образов
- **Ограничивайте ресурсы для pods**
  ставьте requests/limits

КАК АДМИНИСТРАТОР КЛАСТЕРА
усиливаем кластер и контроль

- **Настройте default-deny NetworkPolicy**
  блокируйте весь трафик по умолчанию
- **Используйте least-privilege RBAC**
  давайте минимально необходимые права
- **Шифруйте секреты в покое**
  включите encryption (например, KMS)
- **Держите control plane в приватной сети**
  ограничьте доступ и включите audit логирование
- **Включите admission control**
  используйте политику проверок (OPA, Kyverno и др.)

