libvalidations.sh — встроенные валидации для НАЙС.ОС Container

libvalidations.sh — встроенные валидации для НАЙС.ОС Container

Единая библиотека проверок (валидаторов) для bash-скриптов и приложений в официальных образах НАЙС.ОС Container. Минимум зависимостей, предсказуемые коды возврата, понятные русские логи через `liblog.sh` и безопасные дефолты.

Где находится: библиотека уже предустановлена в образах НАЙС.ОС *Container* и доступна по пути `/nicesoft/niceos/scripts/libvalidations.sh`. Копировать её не нужно — достаточно подключить командой:
`. /nicesoft/niceos/scripts/libvalidations.sh`

1. Назначение и основные принципы

- **Единство:** одинаковые проверки во всех скриптах/сервисах, одинаковые сообщения об ошибках.
- **Минимализм:** не меняет ваши `set`/`shopt`; зависит только от стандартных утилит GNU.
- **Чёткие контракты:** *0* — проверка пройдена; не-0 — проверка провалена (со смысловыми кодами там, где уместно).
- **Интеграция с логированием:** если доступен `liblog.sh`, все сообщения уходят через него; если нет — тихий безопасный фолбэк в `stderr`.
- **RU-сообщения:** все ошибки и предупреждения — на русском языке, без i18n-слоёв.

2. Быстрый старт

```
#!/usr/bin/env bash
set -Eeuo pipefail

# Подключаем валидации (liblog подхватится автоматически, если установлен)
. /nicesoft/niceos/scripts/libvalidations.sh

# Примеры простых проверок
validate_port 8080            || exit 1
validate_ipv4 10.0.0.5        || exit 1
validate_hostname example-host  || exit 1
validate_url "https://example.org/api?v=1" || exit 1

# Парсинг "удобных" значений
BYTES=$(parse_size_bytes 256MiB)       || exit 1
SECS=$(parse_duration_secs 1h30m)      || exit 1
BOOL=$(parse_bool yes)                 || exit 1

info "Проверки пройдены. bytes=${BYTES}, secs=${SECS}, bool=${BOOL}"
```

Все функции возвращают код `0` при успешной проверке и `не-0` — при ошибке. Сообщения об ошибке печатаются в `stderr` (через `liblog` или встроенный фолбэк).

3. Обзор функций

Группа	Функция	Назначение	Пример
Числа	`is_int`	Проверка целого (знак допускается)	`is_int "-42"`
Числа	`is_positive_int`	Неотрицательное целое	`is_positive_int "0"`
Булево	`is_boolean_yes`	`1|yes|true`	`is_boolean_yes yes`
Булево	`is_yes_no_value`	`yes|no`	`is_yes_no_value no`
Булево	`is_true_false_value`	`true|false`	`is_true_false_value true`
Булево	`is_1_0_value`	`1|0`	`is_1_0_value 1`
Булево	`parse_bool`	Нормализация: возвращает `true`/`false`	`parse_bool on`
Булево	`is_empty_value`	Проверка пустого значения	`is_empty_value "$X"`
Порт	`validate_port`	Диапазон 0..65535; флаг `-unprivileged`	`validate_port -unprivileged 8080`
Сети	`validate_ipv4`, `validate_ipv6`, `validate_ip`	Проверка IPv4/IPv6/любого IP	`validate_ip 2001:db8::1`

Группа	Функция	Назначение	Пример
CIDR	validate_cidr4, validate_cidr6, validate_cidr	Проверка префиксов сети	validate_cidr 10.0.0.0/24
Строки	validate_string	Длина/шаблон/печатаемость/пробелы	validate_string -min-length 3 -no-space -- "abc"
Имена	validate_hostname, validate_fqdn	RFC1123 (hostname/FQDN)	validate_fqdn node01.example.org
Парсинг	parse_size_bytes	Число байт из 10M|10MiB|...	parse_size_bytes 256MiB
Парсинг	parse_duration_secs	Секунды из 1h30m|45m|10|...	parse_duration_secs 2d3h
UUID	validate_uuid	UUID v1/2/3/4/5/7 или any	validate_uuid -v 4 550e8400-e29b-41d4-a716-446655440000
URL	validate_url	http/https; host=FQDN|IPv4|[IPv6]; port/path/query/fragment	validate_url "https://[2001:db8::1]:8443/a?x=1#y"
Email	validate_email	Упрощённый, практичный формат	validate_email user.name+tag@example.org
Пути	validate_path_sane	ABS/REL; тип; права; симлинки	validate_path_sane -abs -must-exist -type dir /data
ENV	require_env, require_nonempty_env	Контроль наличия/непустоты переменных окружения	require_nonempty_env DB_PASSWORD

4. Интеграция с liblog.sh и поведение по умолчанию

Библиотека пытается подключить /nicesoft/niceos/scripts/liblog.sh. Если файл доступен, все сообщения идут через info/warn/error/debug. Если нет — применяется встроенный «тихий» фолбэк на stderr с ISO-временем и уровнями. На функциональность валидаций это не влияет.

```
# Явно включим модуль для наглядности заголовка логов
export NICEOS_MODULE=validator
. /nicesoft/niceos/scripts/libvalidations.sh

validate_port -unprivileged 8080 && info "порт валиден" || exit 1
```

5. Детальные примеры использования

5.1. Проверка конфигурации сервиса

```
#!/usr/bin/env bash
set -Eeuo pipefail
. /nicesoft/niceos/scripts/libvalidations.sh

# Требуем обязательные переменные
require_nonempty_env APP_LISTEN
require_env APP_DEBUG || true  # допускаем пустое значение (например, "")

# ENDPOINT может быть URL или HOST:PORT
if [[ "$APP_LISTEN" =~ ^https?:// ]]; then
  validate_url "$APP_LISTEN" || exit 1
else
  HOST=${APP_LISTEN%%:*}
  PORT=${APP_LISTEN##*:}
  validate_hostname "$HOST" || validate_ip "$HOST" || exit 1
  validate_port -unprivileged "$PORT" || exit 1
fi

# Разбор удобных человекочитаемых значений
MAX_SIZE=${APP_MAX_SIZE:-256MiB}
TIMEOUT=${APP_TIMEOUT:-1m}
BYTES=$(parse_size_bytes "$MAX_SIZE") || exit 1
SECS=$(parse_duration_secs "$TIMEOUT") || exit 1

info "Конфигурация валидна: bytes=${BYTES}, timeout=${SECS}s"
```

5.2. Валидация путей и прав

```
validate_path_sane -abs -must-exist -type dir -readable /data || exit 1
validate_path_sane -abs -must-exist -type file -readable -no-symlink /etc/app/config.yml || exit 1
```

5.3. Фильтрация пользовательского ввода

```
# Имя пользователя: 3..32, латиница/цифры/дефис, без пробелов и управляющих
validate_string -min-length 3 -max-length 32 -pattern '^[A-Za-z0-9-]+$' -no-control -no-space --
"$USERNAME" || exit 1
```

5.4. UUID и Email

```
validate_uuid -v 4 "$REQUEST_ID" || exit 1
```

```
validate_email "$ADMIN_EMAIL" || exit 1
```

5.5. Комбинированные проверки с отчётливой диагностикой

```
check_all(){
 . /nicesoft/niceos/scripts/libvalidations.sh
 validate_fqdn "$FQDN" || { error "некорректный FQDN: $FQDN"; return 1; }
 validate_cidr "${NETWORK_CIDR}" || { error "некорректный CIDR: ${NETWORK_CIDR}"; return 1; }
 info "все проверки пройдены"
}
check_all || exit 1
```

6. Использование в Dockerfile и entrypoint

6.1. Базовый образ с проверками на старте

```
FROM niceos/container:latest
ENV NICEOS_MODULE=validator
COPY entrypoint.sh /usr/local/bin/
ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]
```

6.2. Пример `entrypoint.sh`

```
#!/usr/bin/env bash
set -Eeuo pipefail
. /nicesoft/niceos/scripts/libvalidations.sh

require_nonempty_env APP_PORT
validate_port -unprivileged "$APP_PORT" || exit 1
validate_path_sane -abs -must-exist -type dir -writable /data || exit 1

info "Входные параметры корректны, запускаю приложение"
exec /usr/local/bin/myapp --port "$APP_PORT"
```

7. Коды возврата и обработка ошибок

Почти все функции используют простое правило: `0` — успех, `1` — общая ошибка

валидации, другие коды — уточняют причину (например, для портов: *2* — вне диапазона, *3* — привилегированный порт при `-unprivileged`).

```
if ! validate_port -unprivileged "$PORT"; then
  rc=$?
  case $rc in
    2) error "порт вне диапазона или некорректен: $PORT" ;;
    3) error "запрошен привилегированный порт: $PORT" ;;
    *) error "неверный порт: $PORT" ;;
  esac
  exit $rc
fi
```

Все диагностические сообщения печатаются в `stderr`. В скриптах CI/CD удобно комбинировать проверку с явным `exit` и человекочитаемым пояснением.

8. Замечания по безопасности

- Не передавайте секреты напрямую в логи. Формулируйте сообщения нейтрально (например, «пароль пуст», без значения).
- Функции проверки строк могут отсеивать управляющие и непечатаемые символы — используйте флаги `-no-control`/`-printable`.
- Для путей используйте `-no-symlink`, если важно исключить обход через символьные ссылки.
- В URL и Email выполняются здравые (практичные) проверки; для строгих RFC-кейсов добавляйте доменные правила своего приложения.

9. Советы по производительности и эксплуатации

- Валидации реализованы на чистом Bash и стандартных утилитах — накладные расходы минимальны.
- При массовых циклах проверок группируйте их и выводите сводное сообщение, чтобы не «шуметь» в логах.
- Для длительных скриптов используйте `parse_duration_secs` и планируйте *timeouts* разумно.
- При большом количестве входных параметров выносите правила в отдельные функции «политики валидации».

10. Частые рецепты

10.1. Проверка «HOST:PORT»

```
check_hostport(){
 local hp="$1" host port
 [[ "$hp" == *:* ]] || { error "ожидался HOST:PORT"; return 1; }
 host=${hp%%:*}; port=${hp##*:}
 validate_hostname "$host" || validate_ip "$host" || return 1
 validate_port -unprivileged "$port" || return 1
}
check_hostport "$ENDPOINT" || exit 1
```

10.2. Нормализация флагов включения/выключения

```
ENABLED=$(parse_bool "${APP_ENABLED:-false}") || exit 1
if [ "$ENABLED" = "true" ]; then info "функция включена"; else info "функция выключена"; fi
```

10.3. Проверка абсолютного файла с правами

```
validate_path_sane -abs -must-exist -type file -readable -no-symlink -- "$CONFIG" || exit 1
```

10.4. Принятие только печатаемых строк без пробелов

```
validate_string -min-length 1 -max-length 128 -printable -no-space -- "$TOKEN" || exit 1
```

libvalidations.sh в Kubernetes/Helm — практические примеры и прод-чеклист

Дополнение к документации *libvalidations.sh*: как грамотно применять валидации в манифестах Kubernetes и чартах Helm, и на что обратить внимание в продакшене при приёме параметров.

1. Общие принципы интеграции в Kubernetes

- **Единая точка проверки:** все критичные пользовательские параметры (порт, адреса, пути, таймауты, размеры) валидируйте в entrypoint до запуска основного процесса.
- **Fail fast:** при ошибке конфигурации контейнер должен завершиться с понятным русским сообщением (stderr), чтобы `kubectl logs` подсказал, что исправить.
- **Чистые логи:** используйте `liblog.sh` вместе с `libvalidations.sh`, чтобы сообщения были единообразны и легко парсились.
- **Secrets/Config:** принимайте значения через `ConfigMap`/`Secret` → env и валидируйте, не выводя секреты в логи.
- **Сигналы K8s:** корректно завершайте работу (trap ERR/TERM) и пишите причину в лог.

2. Подготовка контейнера (Dockerfile) для Kubernetes

Достаточно наследоваться от базового образа НАЙС.ОС Container: библиотека уже предустановлена.

```
FROM niceos/container:latest
ENV NICEOS_MODULE=app
COPY entrypoint.sh /usr/local/bin/
ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]
```

Далее при деплое передавайте значения через переменные окружения (`env:`), см. примеры ниже.

3. Пример Deployment с валидацией env-параметров

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-app
spec:
  replicas: 1
  selector:
    matchLabels: { app: demo-app }
  template:
    metadata:
      labels: { app: demo-app }
    spec:
```

```
    containers:
      - name: demo
        image: registry.local/demo:latest
        env:
          - name: APP_LISTEN
            value: "0.0.0.0:8080"    # либо https://example.org:8443
          - name: APP_TIMEOUT
            value: "1m30s"         # человекочитаемо
          - name: APP_MAX_SIZE
            value: "256MiB"
          - name: ADMIN_EMAIL
            valueFrom:
              secretKeyRef: { name: app-secret, key: admin_email }
        ports:
          - containerPort: 8080
        readinessProbe:
          exec:
            command: ["/usr/local/bin/entrypoint.sh", "--probe=ready"]
          initialDelaySeconds: 5
          periodSeconds: 10
        livenessProbe:
          exec:
            command: ["/usr/local/bin/entrypoint.sh", "--probe=live"]
          initialDelaySeconds: 10
          periodSeconds: 20
```

В этом примере readiness/liveness используют тот же entrypoint, который выполняет базовые проверки без запуска «тяжёлого» приложения.

4. Пример entrypoint с режимами: запуск и пробы

```
#!/usr/bin/env bash
set -Eeuo pipefail
. /nicesoft/niceos/scripts/liblog.sh
. /nicesoft/niceos/scripts/libvalidations.sh
export NICEOS_MODULE="entrypoint"

handle_probe(){
  case "${1:-}" in
    --probe=ready)
      # Лёгкие проверки окружения
      require_nonempty_env APP_LISTEN || exit 1
      info "readiness: конфигурация доступна"
      exit 0
      ;;
    --probe=live)
      # Простая проверка доступности PID/сокета/порта
      info "liveness: контейнер жив"
      exit 0
```

```
    ;;
  esac
}

handle_probe "${1:-}" || true

# Полная валидация параметров
require_nonempty_env APP_LISTEN
TIMEOUT=$(parse_duration_secs "${APP_TIMEOUT:-1m}") || exit 1
BYTES=$(parse_size_bytes "${APP_MAX_SIZE:-256MiB}") || exit 1

if [[ "$APP_LISTEN" =~ ^https?:// ]]; then
  validate_url "$APP_LISTEN" || exit 1
else
  host=${APP_LISTEN%%:*}; port=${APP_LISTEN##*:}
  validate_hostname "$host" || validate_ip "$host" || exit 1
  validate_port -unprivileged "$port" || exit 1
fi

if [ -n "${ADMIN_EMAIL:-}" ]; then
  validate_email "$ADMIN_EMAIL" || exit 1
fi

info "Запуск приложения: listen=${APP_LISTEN} timeout=${TIMEOUT}s max_size=${BYTES}B"
exec /usr/local/bin/myapp "$@"
```

5. Helm: шаблоны значений и безопасные дефолты

Задавайте осмысленные дефолты в `values.yaml` и используйте в шаблонах только явно документированные параметры.

5.1. values.yaml (пример)

```
replicaCount: 1
image:
  repository: registry.local/demo
  tag: latest
  pullPolicy: IfNotPresent
app:
  listen: "0.0.0.0:8080"      # или https://example.org:8443
  timeout: "90s"              # человекочитаемо
  maxSize: "256MiB"
  adminEmail: ""
resources: {}
nodeSelector: {}
tolerations: []
```

```
affinity: {}
```

5.2. deployment.yaml (фрагмент шаблона)

```
env:
 - name: APP_LISTEN
   value: {{ .Values.app.listen | quote }}
 - name: APP_TIMEOUT
   value: {{ .Values.app.timeout | quote }}
 - name: APP_MAX_SIZE
   value: {{ .Values.app.maxSize | quote }}
 - name: ADMIN_EMAIL
   value: {{ .Values.app.adminEmail | quote }}
```

Секреты храните в `Secret` и прокидывайте через `valueFrom.secretKeyRef`. Значения, вводимые пользователем, дополнительно валидируйте в `entrypoint.sh`.

6. Pre-install Hook: «сухая» проверка значений

Можно добавить «hook»-Job для быстрой проверки критичных параметров ещё до запуска основного Pod.

```
apiVersion: batch/v1
kind: Job
metadata:
 name: {{ include "demo.fullname" . }}-precheck
 annotations:
   "helm.sh/hook": pre-install,pre-upgrade
   "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded
spec:
 template:
   spec:
     restartPolicy: Never
     containers:
       - name: precheck
         image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
         command: ["/bin/bash","-lc","/usr/local/bin/precheck.sh"]
         env:
           - name: APP_LISTEN
             value: {{ .Values.app.listen | quote }}
           - name: APP_TIMEOUT
             value: {{ .Values.app.timeout | quote }}
           - name: APP_MAX_SIZE
             value: {{ .Values.app.maxSize | quote }}
```

Содержимое `precheck.sh`

```
#!/usr/bin/env bash
set -Eeuo pipefail
. /nicesoft/niceos/scripts/liblog.sh
. /nicesoft/niceos/scripts/libvalidations.sh
export NICEOS_MODULE="precheck"

require_nonempty_env APP_LISTEN
parse_duration_secs "${APP_TIMEOUT:-1m}" >/dev/null
parse_size_bytes    "${APP_MAX_SIZE:-256MiB}" >/dev/null

if [[ "$APP_LISTEN" =~ ^https?:// ]]; then
  validate_url "$APP_LISTEN"
else
  host=${APP_LISTEN%%:*}; port=${APP_LISTEN##*:}
  validate_hostname "$host" || validate_ip "$host"
  validate_port -unprivileged "$port"
fi

info "precheck: параметры выглядят валидными"
```

Если precheck завершится с ошибкой, Helm остановит установку с понятным сообщением в логах hook-Joba.

7. Sidecar/InitContainer: подготовка каталогов и проверка путей

Для подготовки директорий (права, отсутствие симлинков) используйте `initContainers` и `validate_path_sane`.

```
initContainers:
  - name: prepare-data
    image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
    command: ["/bin/bash","-lc","/usr/local/bin/init-prepare.sh"]
    volumeMounts:
      - name: data
        mountPath: /data
```

init-prepare.sh

```
#!/usr/bin/env bash
set -Eeuo pipefail
. /nicesoft/niceos/scripts/libvalidations.sh

validate_path_sane -abs -must-exist -type dir -writable -no-symlink /data
```

8. Чек-лист приёма параметров в продакшене

1. **Документируйте** каждую переменную окружения/значение Helm: назначение, тип, допустимый диапазон, дефолт.
2. **Валидируйте** все критичные параметры в entrypoint: порт, адреса, URL, FQDN, CIDR, пути (тип/права), размеры, длительности, email, UUID.
3. **Не логируйте секреты**. Сообщения формулируйте нейтрально: «секрет пуст/отсутствует», без значений.
4. **Fail fast**: при ошибке завершайтесь с ненулевым кодом и понятным сообщением на русском.
5. **Осознанные дефолты**: устанавливайте безопасные значения в `values.yaml`; явно указывайте единицы (MiB, s, m, h).
6. **Rate-limit логов**: если ожидается бурст предупреждений, используйте возможности `liblog.sh` для склейки повторов.
7. **Пути**: запрещайте `..`, управляющие символы, симлинки там, где это критично (`-no-symlink`).
8. **Сети**: проверяйте IP/CIDR/порты; для публичных URL используйте `validate_url` и явные порты.
9. **Readiness/Liveness**: пробы должны быть быстрыми и не раскрывать чувствительных деталей в логах.
10. **Наблюдаемость**: включайте `NICEOS_MODULE` для чёткого источника логов, используйте `json` формат при интеграции с лог-агрегаторами.
11. **Производительность**: избегайте дорогостоящих внешних вызовов в валидаторах; все проверки — локальные.
12. **Тестируемость**: добавьте pre-install hook (Helm Job) для «сухой» проверки значений до развёртывания.
13. **Бэкапы/ротация**: если пишете в файл, позаботьтесь о ротации и персистентности тома.
14. **Совместимость**: держите Bash-скрипты POSIX-совместимыми там, где возможно; избегайте GNU-специфики вне контейнера.

Итого: строгая валидация на старте + понятные логи = быстрые откаты и минимальные простои.

libvalidations.sh в экосистеме Kubernetes/Helm — полное дополнение

Шаблоны ConfigMap/Secret, best practices для Helm, pre/post hooks, Helm tests, а также готовые сниппеты интеграции логов с Fluent Bit и Vector. Всё — в одном месте.

Это дополнение продолжает предыдущие материалы по **libvalidations.sh** и ориентировано на прод-использование в Kubernetes и Helm.

1. Values, ConfigMap и Secret: безопасный конвейер параметров

Рекомендуемая схема передачи значений: `values.yaml` → `ConfigMap/Secret` → `env` → `entrypoint` с валидацией через `libvalidations.sh`.

1.1. values.yaml (пример)

```
app:
  listen: "0.0.0.0:8080"      # или https://example.org:8443
  timeout: "90s"              # человекочитаемый формат
  maxSize: "256MiB"
  adminEmail: ""
  # Для секретов не храните значения в values.yaml — используйте отдельные Secret values или внешние секрет-менеджеры
```

1.2. ConfigMap (шаблон)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ include "app.fullname" . }}-config
  labels:
    app.kubernetes.io/name: {{ include "app.name" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
```

```
    helm.sh/chart: {{ include "app.chart" . }}
data:
  APP_LISTEN: {{ .Values.app.listen | quote }}
  APP_TIMEOUT: {{ .Values.app.timeout | quote }}
  APP_MAX_SIZE: {{ .Values.app.maxSize | quote }}
```

1.3. Secret (шаблон)

Секреты — только в Secret. Никогда не логируйте их содержимое, валидируйте лишь факт наличия/непустоты.

```
apiVersion: v1
kind: Secret
metadata:
  name: {{ include "app.fullname" . }}-secret
  labels:
    app.kubernetes.io/name: {{ include "app.name" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
    helm.sh/chart: {{ include "app.chart" . }}
type: Opaque
data:
  ADMIN_EMAIL: {{ .Values.app.adminEmail | b64enc | quote }}
```

1.4. Подключение в Deployment (envFrom/env)

```
envFrom:
  - configMapRef: { name: {{ include "app.fullname" . }}-config }
  - secretRef:    { name: {{ include "app.fullname" . }}-secret }
# Либо точечно через env:
# env:
#   - name: APP_LISTEN
#     valueFrom: { configMapKeyRef: { name: {{ include "app.fullname" . }}-config, key: APP_LISTEN } }
#   - name: ADMIN_EMAIL
#     valueFrom: { secretKeyRef: { name: {{ include "app.fullname" . }}-secret, key: ADMIN_EMAIL } }
```

Важно: в `entrypoint.sh` используйте `require_nonempty_env` для обязательных значений без вывода самих секретов в лог.

2. Helm: schema-валидация values и защитные

дефолты

Helm поддерживает проверку схемы через `values.schema.json`. Это позволяет сразу отсеивать некорректные типы/диапазоны ещё на этапе рендеринга чартов.

2.1. values.schema.json (фрагмент)

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "type": "object",
 "properties": {
  "app": {
   "type": "object",
   "properties": {
    "listen": { "type": "string", "minLength": 1 },
    "timeout": { "type": "string", "pattern": "^[0-9]+(s|m|h|d)([0-9]+(s|m|h|d))*$" },
    "maxSize": { "type": "string", "pattern": "^[0-9]+(Ki?B|Mi?B|Gi?B|Ti?B|[KMGTP]i?B?|B)?$" },
    "adminEmail": { "type": "string" }
   },
   "required": ["listen"],
   "additionalProperties": false
  }
 }
}
```

Схема не заменяет `libvalidations.sh`, но позволяет отлавливать типовые ошибки ещё до деплоя.

3. Pre-install/Pre-upgrade hooks: «сухая» проверка перед релизом

Добавьте Job-hook, который запустит «precheck» скрипт и отвалидирует критичные параметры. Если что-то не так — установка/обновление остановится с понятным сообщением.

```
apiVersion: batch/v1
kind: Job
metadata:
 name: {{ include "app.fullname" . }}-precheck
 annotations:
  "helm.sh/hook": pre-install,pre-upgrade
  "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded
spec:
```

```
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: precheck
          image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
          command: ["/bin/bash","-lc","/usr/local/bin/precheck.sh"]
          envFrom:
            - configMapRef: { name: {{ include "app.fullname" . }}-config }
            - secretRef:    { name: {{ include "app.fullname" . }}-secret }
```

Содержимое `precheck.sh`

```
#!/usr/bin/env bash
set -Eeuo pipefail
. /nicesoft/niceos/scripts/liblog.sh
. /nicesoft/niceos/scripts/libvalidations.sh
export NICEOS_MODULE="precheck"

require_nonempty_env APP_LISTEN
parse_duration_secs "${APP_TIMEOUT:-1m}" >/dev/null
parse_size_bytes    "${APP_MAX_SIZE:-256MiB}" >/dev/null

if [[ "$APP_LISTEN" =~ ^https?:// ]]; then
  validate_url "$APP_LISTEN"
else
  host=${APP_LISTEN%%:*}; port=${APP_LISTEN##*:}
  validate_hostname "$host" || validate_ip "$host"
  validate_port -unprivileged "$port"
fi

info "precheck: параметры выглядят валидными"
```

4. Helm tests: пост-проверки после релиза

Helm tests — это Jobs, запускаемые командой `helm test <release>` после установки/обновления. Они могут убедиться, что сервис поднялся и принял конфигурацию.

```
apiVersion: v1
kind: Pod
metadata:
  name: {{ include "app.fullname" . }}-test
  annotations:
    "helm.sh/hook": test
spec:
```

```
  restartPolicy: Never
  containers:
    - name: smoke
      image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
      command: ["/bin/bash","-lc"]
      args:
        - |
          set -Eeuo pipefail
          . /nicesoft/niceos/scripts/liblog.sh
          . /nicesoft/niceos/scripts/libvalidations.sh
          export NICEOS_MODULE=test
          # Пример: проверим, что endpoint отвечает и порт валиден из values
          validate_port -unprivileged {{ (split ":" .Values.app.listen)._1 | default 8080 }}
          # Лёгкий HTTP-запрос (если образ содержит curl)
          if command -v curl &>/dev/null; then
            curl -fsS http://{{ include "app.fullname" . }}:{{ (split ":" .Values.app.listen)._1 | default 8080
}}/healthz
          fi
          info "helm test: ok"
```

Подсказка: для аккуратного парсинга порта из `listen` проще разнести host/port в отдельные поля values. Тогда и схема, и шаблоны будут чище.

5. Readiness/Liveness/Startup probes: согласованность с валидациями

- **readiness**: проверяет готовность принимать трафик (быстрые локальные проверки окружения/сокетов).
- **liveness**: убедиться, что процесс не завис; простые проверки без тяжёлых запросов.
- **startup**: полезна, если приложение долго «прогревается». На время прогрева *liveness* лучше отключать.

```
readinessProbe:
  exec: { command: ["/usr/local/bin/entrypoint.sh", "--probe=ready"] }
livenessProbe:
  exec: { command: ["/usr/local/bin/entrypoint.sh", "--probe=live"] }
startupProbe:
  exec: { command: ["/usr/local/bin/entrypoint.sh", "--probe=start"] }
  periodSeconds: 10
  failureThreshold: 30
```

6. Логи: Fluent Bit и Vector (сниппеты)

Благодаря `liblog.sh` логи уже структурированы и пригодны для перехвата. Ниже — минимальные конфигурации для двух популярных агентов.

6.1. Fluent Bit: сбор из stdout

```
[SERVICE]
    Daemon       Off
    Log_Level    info

[INPUT]
    Name         tail
    Path         /var/log/containers/*.log
    Parser       docker
    Tag          kube.*

[FILTER]
    Name         kubernetes
    Match        kube.*

# Если вы используете JSON-формат (NICEOS_LOG_FORMAT=json),
# добавьте парсер и роут в output по ключам level/module/msg.

[OUTPUT]
    Name         stdout
    Match        *
```

6.2. Vector: сбор stdout и маршрутизация

```
[sources.kube]
  type = "kubernetes_logs"

[transforms.only_app]
  type = "filter"
  inputs = ["kube"]
  condition = ".kubernetes.container_name == \"app\""

# Если используете NICEOS_LOG_FORMAT = "json":
# можно выделять поля и отправлять в системное хранилище
[transforms.parse_json]
  type = "remap"
  inputs = ["only_app"]
  source = """
    . = parse_json!(.message)
  """
```

```
[outputs.dev]
  type = "console"
  inputs = ["parse_json"]
  encoding.codec = "json"
```

Для файловых логов (режим `both` с записью в файл) укажите соответствующие пути в *input* агентов. Помните, что в файл **никогда** не пишутся ANSI-цвета.

7. Прод-чеклист приёма параметров (расширенный)

1. **Контракты:** документируйте каждую переменную/значение Helm (тип, диапазон, дефолт, пример, обязательность).
2. **Двухступенчатая проверка:** используйте `values.schema.json` + валидацию на старте контейнера через `libvalidations.sh`.
3. **Никаких секретов в логах:** только факт наличия/пустоты; маскируйте потенциальные секреты в сообщениях.
4. **Fail fast:** при ошибке параметров — немедленный `exit > 0` и понятный текст на русском в `stderr`.
5. **Единицы и формат:** всегда указывайте единицы (`MiB`, `s`, `m`, `h`); принимайте человекочитаемые значения (`parse_size_bytes`, `parse_duration_secs`).
6. **Сетевые параметры:** `validate_ip`/`validate_cidr`/`validate_port`; запрещайте привилегированные порты при необходимости.
7. **Пути и права:** `validate_path_sane` с флагами `-abs`/`-must-exist`/`-no-symlink` и проверками прав.
8. **Чистые логи:** используйте формат `json` для агрегации; при бурстах — склейку повторов из `liblog.sh`.
9. **Наблюдаемость:** задавайте `NICEOS_MODULE`; для разных контейнеров — уникальные значения (web/api/worker).
10. **Проверки после релиза:** добавьте Helm tests, которые делают простые smoke-проверки.
11. **Готовность/живость:** пробы делайте быстрыми и неразговорчивыми; не раскрывайте конфиденциальные детали.
12. **Документация для пользователей чарта:** README с примерами values, таблицей переменных и ожидаемым поведением при ошибках.

Helm README (таблица значений), StatefulSet и прод-паттерны HPA/PDB

Полноценное дополнение к материалам о **libvalidations.sh** и интеграции в Kubernetes/Helm: готовый каркас README для чарта, примеры StatefulSet и настройки отказоустойчивости через HPA и PDB.

1) Каркас README для Helm-чарта

Ниже — опорный README, который можно положить в корень чарта (`charts/<name>/README.md`). Он описывает назначение, установку, апгрейд, значения и их валидацию. Таблица значений синхронизируется с `values.yaml` и `values.schema.json`.

README.md (шаблон)

```
# <app-name> Helm Chart

Официальный чарт приложения <app-name> для Kubernetes. Использует встроенные библиотеки НАЙС.ОС Container: `liblog.sh` и `libvalidations.sh`.

## Установка

```bash
helm repo add <repo> <url>
helm upgrade --install app <repo>/<chart> \
  --namespace app --create-namespace \
  -f values.yaml
```

Обновление

```bash
helm upgrade app <repo>/<chart> -f values.yaml
```

Переменные (Values)

Ключ	Тип	По умолчанию	Описание
`replicaCount`	int	`1`	Количество реплик Pod
`image.repository`	string	`registry.local/demo`	Репозиторий образа
`image.tag`	string	`latest`	Тег образа
`image.pullPolicy`	string	`IfNotPresent`	Политика загрузки
```

`app.listen`	string	`"0.0.0.0:8080"`	Адрес прослушивания (`HOST:PORT` или `http(s)://host[:port]`)
`app.timeout`	string	`"90s"`	Тайм-аут (поддерживает `Xs`, `Xm`, `Xh`, `Xd`)
`app.maxSize`	string	`"256MiB"`	Макс. размер (поддерживает `KiB/MiB/GiB/TiB`)
`app.adminEmail`	string	`""`	E-mail администратора (может быть пустым)
`resources`	object	`{}`	Лимиты/запросы CPU и RAM
`nodeSelector`	map	`{}`	Распределение по нодам
`tolerations`	list	`[]`	Толерации
`affinity`	map	`{}`	Правила аффинити
`hpa.enabled`	bool	`false`	Включить горизонтальное авто-масштабирование
`hpa.minReplicas`	int	`2`	Минимум реплик (если включено HPA)
`hpa.maxReplicas`	int	`10`	Максимум реплик
`hpa.cpu.targetAverageUtilization`	int	`70`	Целевая загрузка CPU (%)
`pdb.enabled`	bool	`true`	Включить PodDisruptionBudget
`pdb.minAvailable`	string	`"50%"`	Мин. доступных Pod при эвакуациях

Валидация значений

На уровне Helm: `values.schema.json` — типы/паттерны.

На уровне контейнера: `entrypoint.sh` использует `libvalidations.sh` и завершится с ошибкой (fail-fast) при некорректных параметрах.

Мониторинг и логи

Рекомендуется `NICEOS_LOG_FORMAT=json` для интеграции с лог-агентами. Повторы сообщений агрегируются средствами `liblog.sh`.

Лицензия

© 2025, ООО «НАЙС СОФТ». Проприетарное ПО.

2) Пример StatefulSet + Service

StatefulSet пригодится для stateful-сервисов: стабильные имена Pod, персистентные тома и упорядоченный запуск/останов.

statefulset.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: demo-sts
spec:
  serviceName: demo-headless
  replicas: 3
  selector:
```

```
    matchLabels: { app: demo-sts }
  template:
    metadata:
      labels: { app: demo-sts }
    spec:
      containers:
        - name: app
          image: registry.local/demo:latest
          env:
            - name: APP_LISTEN
              value: "0.0.0.0:8080"
          ports:
            - name: http
              containerPort: 8080
          volumeMounts:
            - name: data
              mountPath: /var/lib/app
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes: [ "ReadWriteOnce" ]
        resources:
          requests:
            storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: demo-headless
spec:
  clusterIP: None
  selector:
    app: demo-sts
  ports:
    - name: http
      port: 8080
      targetPort: 8080
```

В entrypoint используйте `libvalidations.sh` для проверки путей и прав: `validate_path_sane -abs -must-exist -type dir -writable /var/lib/app`.

3) HPA (горизонтальное авто-масштабирование)

Классический пример HPA по CPU. Параметры обычно выносите в `values.yaml` и включаете через флаг `hpa.enabled`.

hpa.yaml

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: demo-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: demo-app
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Для расширенных сценариев добавьте метрики по памяти и пользовательские метрики (Prometheus Adapter). Убедитесь, что probes быстры и не создают лишнюю нагрузку.

4) PDB (PodDisruptionBudget) — защита от одновременных эвакуаций

PDB ограничивает добровольные прерывания (evictions), сохраняя минимальное число работающих Pod при обновлениях узлов и кластера.

pdb.yaml

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: demo-pdb
spec:
  minAvailable: 50%
  selector:
    matchLabels:
      app: demo-app
```

Если используете StatefulSet/Deployment с небольшим количеством реплик (например, 2), планируйте значения `minAvailable`/`maxUnavailable` так, чтобы не блокировать обновления кластера.

5) Встраивание HPA/PDB в Helm-чарт

Ниже — фрагменты шаблонов с условиями включения.

templates/hpa.yaml

```
{{- if .Values.hpa.enabled }}
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: {{ include "app.fullname" . }}
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: {{ include "app.fullname" . }}
  minReplicas: {{ .Values.hpa.minReplicas }}
  maxReplicas: {{ .Values.hpa.maxReplicas }}
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: {{ .Values.hpa.cpu.targetAverageUtilization }}
{{- end }}
```

templates/pdb.yaml

```
{{- if .Values.pdb.enabled }}
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: {{ include "app.fullname" . }}-pdb
spec:
  minAvailable: {{ .Values.pdb.minAvailable | quote }}
  selector:
    matchLabels:
      app.kubernetes.io/name: {{ include "app.name" . }}
      app.kubernetes.io/instance: {{ .Release.Name }}
```

```
{{- end }}
```

6) Рекомендации по прод-эксплуатации

- **Валидация на каждом слое:** JSON-схема values + `libvalidations.sh` в entrypoint + пробные запуск/тесты (Helm tests).
- **Запасы по ресурсам:** задавайте Requests/Limits и проектируйте HPA так, чтобы не «душить» холодные старты.
- **Высокая доступность:** PDB + ≥2 реплик, корректные probes, разнесение по зонам (topology spread constraints).
- **Наблюдаемость:** формат логов `json`, поля `module/level/ts`, метрики readiness/failures.
- **Секреты:** только в Secret; в логах — никакой печати значений, только факт наличия/пустоты.
- **Idempotent-деплой:** pre-hooks для «сухих» проверок, post-hooks/Helm tests для smoke-проверок.