Установщик NiceOSInstaller

Введение

Зачем NiceOS потребовался собственный установщик

Проект NiceOS изначально создавался как универсальная защищённая операционная система, ориентированная на автоматическое развёртывание, централизованное управление и соответствие требованиям информационной безопасности. В рамках этих задач традиционные установщики — такие как **Anaconda**, **Calamares** или **Debian Installer** — оказались недостаточно гибкими и масштабируемыми.

NiceOS нуждалась в инструменте, который бы:

- **Поддерживал установку без участия пользователя** (автоматическая инсталляция на основе JSON-конфигураций, пригодных для CI/CD и массового развёртывания);
- **Обеспечивал тонкую настройку структуры разделов**, включая поддержку теневых разделов для A/B-обновлений, шифрования, LVM и специализированных схем разметки;
- **Интегрировался с контейнерами и внешними сборочными системами**, включая подгрузку Docker-образов и выполнение Ansible-плейбуков в процессе установки;
- **Поддерживал различные режимы вывода**: ISO-образы для живых сред и автоустановки, OVA-шаблоны для VMware, AMI-образы для AWS;
- **Позволял писать расширения на Python**, не прибегая к громоздким языкам или сложной архитектуре модульности;
- **Имел минимальные зависимости**, подходя как для графических, так и полностью headless-установок.

Ни один из существующих установщиков не отвечал всем этим требованиям сразу. Anaconda требовал сложной настройки и модификации для headless-режимов, Calamares ориентирован на настольные дистрибутивы с графическим интерфейсом, а Debian Installer был недостаточно гибким в плане container-интеграции и модульной архитектуры.

Таким образом, было принято решение разработать **собственный установщик**, ориентированный прежде всего на автоматизацию, модульность и универсальность. Это позволило не только удовлетворить внутренние потребности NiceOS, но и получить независимый инструмент, пригодный для использования в других проектах.

История проекта (на базе Photon OS Installer)

Разработка NiceOS Installer началась как форк **Photon OS Installer** — минималистичного установщика от VMware, созданного для серверной системы Photon Linux. В оригинале он был нацелен на установку с ISO и поддерживал базовые функции: разметку дисков, установку пакетов через `tdnf`, конфигурацию пользователя и загрузчика GRUB.

NiceOS пошёл дальше, сохранив лишь архитектурные идеи:

- минимализм и **отсутствие GUI**;
- модульность через Python-файлы в каталоге `modules/`;
- использование `tdnf` как основного пакетного менеджера;
- полное управление установкой через **kickstart-файл в формате JSON**.

Далее проект эволюционировал в сторону:

- **поддержки различных типов образов** (ISO, OVA, AMI, RAW);
- **динамической загрузки JSON-конфигурации** с внешних источников, включая `guestinfo` в VMware, HTTP-сервера, параметры ядра (`ks=`) и внешние носители;
- **расширения набора модулей постобработки**, отвечающих за локаль, загрузку SSH-ключей, настройку SELinux, добавление системных агентов и др.;
- **поддержки A/B-разделов** и механизмов атомарных обновлений;
- **возможности запуска в headless-режиме** без графики и пользовательского ввода, а также интеграции с системами CI/CD и `cloud-init`.

Таким образом, NiceOS Installer стал независимым и расширяемым решением, способным не только обеспечивать автоматическую установку, но и собирать образы, пригодные для развёртывания в облаках, контейнерах, виртуальных машинах и физических серверах.

Подготовка к работе

Установка через RPM-пакет

Установщик **NiceOS Installer** распространяется в виде RPM-пакета `niceos-installer.rpm` и устанавливается стандартными средствами пакетного менеджера `dnf` или `tdnf`.

Установка выполняется одной командой:

```
sudo tdnf install niceos-installer
```

Или, если используется локальный файл:

```
sudo rpm -ivh niceos-installer.rpm
```

После установки утилита `niceos-installer` становится доступной в `/usr/bin/` и может быть вызвана напрямую:

```
niceos-installer --image-type iso --config /path/to/ks.json
```

Зависимости

Все необходимые зависимости подтягиваются автоматически при установке пакета. Тем не менее, для ручного запуска вне RPM-окружения потребуются:

Компонент	Назначение
`python3` (3.6)	Интерпретатор Python для выполнения установщика
`tdnf`	Пакетный менеджер, используемый в процессе установки
`grub2`	Загрузчик, устанавливаемый в BIOS/UEFI
`parted`, `mkfs`, `cryptsetup`	Разметка дисков, создание файловых систем, шифрование
`rsync`, `curl`	Передача файлов и загрузка JSON-конфигурации

Установщик можно использовать как из ISO-среды, так и из любой системы с доступом к необходимым утилитам и правами root.

Краткий обзор архитектуры

Основные компоненты

Архитектура **NiceOS Installer** построена по модульному принципу, где каждый этап установки реализован в виде отдельного Python-класса или расширяемого модуля. Основные компоненты:

Файл / Класс	Описание
`main.py`	Точка входа. Обрабатывает аргументы командной строки, загружает конфигурацию и запускает соответствующий класс установщика (`Installer` или `IsoInstaller`).
`Installer`	Основной класс, реализующий установку системы на диск: разметка, установка пакетов, настройка загрузчика и выполнение postinstall-модулей.
`IsoInstaller`	Наследник `Installer`, используемый при установке из ISO-образа. Умеет читать конфигурацию из параметра ядра, с HTTP/USB и из `/run/install.ks`.
`isoBuilder.py`	Модуль для создания ISO, OVA, AMI и других типов образов на основе установленной системы и конфигурации. Используется как внутри CI/CD, так и вручную.

Роль ключевых каталогов

Исходный код установщика структурирован по каталогам с чётким назначением. Это облегчает модификацию, отладку и расширение установщика.

Каталог	Назначение
`modules/`	Плагины и сценарии постустановочной настройки: локализация, настройка root-пароля, SSH-доступ, hostname, интеграция с Ansible, cloud-init и др. Пользователь может добавлять свои модули без изменения ядра.
`docs/`	Техническая документация, включая описание структуры kickstart-файлов (`ks_config.md`), описание работы A/B-разделов, и инструкции по расширению установщика.

Каталог	Назначение
sample_ks/	Готовые примеры JSON-конфигураций для различных сценариев: установка на bare metal, в VM, с контейнерами, с шифрованием и с минимальным набором пакетов.

Такое разделение компонентов позволяет использовать niceos-installer не только как CLI-инструмент, но и как **Python-библиотеку** в сторонних системах автоматизации и оркестрации.

Формат kickstart-конфигурации

Установщик **NiceOS Installer** использует расширяемый **JSON-формат kickstart-конфигураций**, позволяющий полностью управлять установкой: от разметки дисков и списка пакетов до выполнения скриптов и загрузки Docker-образов. Это делает конфигурации читаемыми, машинно-обрабатываемыми и легко интегрируемыми в CI/CD и скрипты автоматического развёртывания.

Ключевые параметры конфигурации

Параметр	Описание
hostname	Имя хоста для установленной системы
password	Пароль пользователя root. Поддерживает как text, так и crypted значения (SHA-512-хеш).
disks	Описание целевого диска установки (например, /dev/sda), может поддерживать множественные устройства в будущем.
partitions	Список разделов с указанием mountpoint, size (в MiB) и filesystem. Если size=0, то используется остаток диска.
packagelist_file	Имя файла с JSON-перечнем пакетов (обычно packages_minimal.json), используемого для установки через tdnf.
additional_packages	Список дополнительных пакетов, устанавливаемых помимо основного списка (например, vim).
postinstall	Массив строк/команд оболочки, выполняемых внутри chroot после установки. Обычно начинается с #!/bin/sh.

Параметр	Описание
public_key	Публичный SSH-ключ для доступа к системе без пароля.
linux_flavor	Имя ядра Linux (используется при генерации grub.cfg), по умолчанию linux.
niceos_docker_image	Имя Docker-образа, который должен быть скачан/распакован в систему после установки.

Пример конфигурационного файла

Пример kickstart-файла из каталога sample_ks/:

```
{
 "hostname": "niceos-machine",
 "password": {
  "crypted": false,
  "text": "changeme"
 },
 "disks": {
  "default": {
   "device": "/dev/sda"
  }
 },
 "partitions": [
  {
   "mountpoint": "/",
   "size": 0,
   "filesystem": "ext4"
  },
  {
   "mountpoint": "/boot",
   "size": 128,
   "filesystem": "ext4"
  },
  {
   "mountpoint": "/root",
   "size": 128,
   "filesystem": "ext4"
  },
  {
   "size": 128,
   "filesystem": "swap"
  }
 ],
 "packagelist_file": "packages_minimal.json",
```

```
  "additional_packages": [
    "vim"
  ],
  "postinstall": [
    "#!/bin/sh",
    "echo \"Hello World\" > /etc/postinstall"
  ],
  "public_key": "<ssh-key-here>",
  "linux_flavor": "linux",
  "niceos_docker_image": "niceos:5.0"
}
```

Дополнительные примеры можно найти в директории `sample_ks/`, в том числе сценарии с шифрованием, LVM, cloud-init, Ansible, overlay-монтированием и кастомными Docker-образами.

Пошаговый процесс установки

Установочный процесс **NiceOS Installer** организован как последовательность модульных шагов, каждый из которых отвечает за конкретную фазу установки. Это позволяет гибко управлять поведением инсталлятора, расширять его и отлаживать при необходимости.

1. Чтение параметров ядра и загрузка конфигурации

При запуске в режиме ISO, установщик ищет JSON-файл конфигурации (kickstart) в следующих источниках:

- Параметр ядра `ks=` (например, `ks=http://server/config.json`)
- Файл `/run/install.ks`, размещённый на внешнем носителе
- Метаданные VMware (`guestinfo.ks`), закодированные в base64

Для нестандартных образов конфигурация указывается явно через `--config`:

```
niceos-installer --image-type iso --config /path/to/ks.json
```

2. Запуск `configure()` и выполнение preinstall

Метод `configure()` инициализирует установку: создаётся рабочая директория, проверяется валидность конфигурации, загружаются preinstall-модули из `m_preinstall.py`

и выполняются скрипты, описанные в секции `preinstall` JSON-файла.

Это позволяет предварительно установить зависимости, подключить сетевые ресурсы или подготовить окружение.

3. Разбивка дисков, создание ФС, монтирование

Блок методов `_prepare_devices()`, `_get_disk_sizes()`, `_calc_size_percentages()`, `_partition_disks()` и `_format_partitions()` отвечает за:

- Разметку указанных дисков (`/dev/sdX`, LVM, GPT/MBR, теневые разделы для A/B)
- Форматирование файловых систем (ext4, xfs, swap, luks)
- Создание точек монтирования и подготовку к chroot-установке

Поддерживаются различные режимы: BIOS/UEFI, LUKS-шифрование, автоматическая генерация `/boot` и загрузочных разделов.

4. Настройка репозиториев, установка пакетов через TDNF

Из JSON-конфигурации читается имя файла со списком пакетов (`packagelist_file`). Устанавливается репозиторная конфигурация и запускается `tdnf install`.

Дополнительные RPM-пакеты (например, из каталога `rpms/`) могут быть установлены вручную через метод `_install_additional_rpms()`.

5. Выполнение Ansible или Docker-образов

При наличии параметров `ansible_playbook` или `niceos_docker_image` установщик запускает соответствующий код:

- **Ansible:** выполнение указанных плейбуков в chroot-окружении
- **Docker:** загрузка и распаковка контейнера в директорию целевой системы

Это позволяет интегрировать готовые профили, роли, агенты мониторинга или DevOps-инструменты в процессе установки.

6. Настройка системы через модули

Внутри chroot запускаются Python-модули из каталога `modules/`, каждый из которых отвечает за определённую настройку:

- `m_updaterootpassword.py` — установка пароля root
- `m_updatesshconfig.py` — добавление SSH-ключей и настройка доступа
- `m_sethostname.py` — установка имени хоста
- `m_locale.py` — локализация, timezone, язык

Пользователь может добавлять собственные модули, не меняя ядро установщика.

7. Установка GRUB и финальные шаги

На последнем этапе:

- Устанавливается загрузчик GRUB в зависимости от режима (`bios`, `efi`, `dualboot`)
- Генерируется `grub.cfg`
- Создаётся `fstab`, копируются ключи, ядро, initrd
- Выполняются постустановочные скрипты из `postinstall` и модуля `m_postinstall.py`
- Очищаются временные каталоги, размонтируются разделы

После завершения инсталляции отображается поздравительное сообщение либо создаётся образ (в случае запуска в режиме генерации ISO/OVA/AMI).

Режимы работы и создание образов

Установщик **NiceOS Installer** поддерживает несколько режимов работы в зависимости от сценария использования. Он может быть запущен как интерактивно (из ISO), так и в автоматическом режиме в рамках CI/CD, создавая готовые образы для виртуальных машин, облаков и физического оборудования.

Установка с ISO (IsoInstaller)

В режиме ISO используется класс `IsoInstaller`, который дополнительно реализует загрузку конфигурации из параметров ядра или внешних источников:

- Загрузка JSON-конфигурации по параметру `ks=`
- Автоматическое определение внешних носителей (например, USB-диск с

/install.ks)
- Загрузка по HTTP/HTTPS
- Чтение из guestinfo.ks при установке в VMware (base64-строка)

Такой режим подходит для автономной установки на физическое оборудование или развёртывания в средах без сети.

При использовании ISO-режима все шаги происходят без необходимости доступа к сети или окружению разработчика — достаточно одной флешки с образом.

Генерация собственных ISO/OVA/AMI при помощи isoBuilder

Утилита isoBuilder.py входит в состав установщика и позволяет формировать образы следующих типов:

- **ISO** — для записи на USB или загрузки в виртуальной машине
- **OVA** — виртуальный шаблон для VMware/VirtualBox
- **AMI** — образ для загрузки в Amazon EC2
- **RAW** — универсальный блочный образ для кастомных решений (например, cloud-init)

Пример команды для генерации ISO:

```
python3 isoBuilder.py --type iso --config ks.json --output niceos.iso
```

Все параметры (включая структуру разделов, список пакетов, hostname и postinstall-скрипты) берутся из указанного JSON-файла. Это позволяет полностью автоматизировать сборку образов под конкретные задачи.

Благодаря isoBuilder тот же код, который используется при обычной установке, применяется и при создании образов. Это гарантирует предсказуемость и повторяемость результатов.

Внутри isoBuilder используется тот же механизм, что и в Installer, с дополнительными этапами упаковки в формат ISO, создания манифестов и настройки загрузчика.

Расширяемость

Одним из ключевых преимуществ **NiceOS Installer** является его **модульная**

архитектура. Установщик проектировался таким образом, чтобы его можно было легко дополнять новыми функциями — без изменения основного кода. Это особенно важно для интеграции с внешними сервисами, кастомизации postinstall-процессов или поддержки специфических требований.

Собственные модули в `photon_installer/modules/`

Каждый модуль — это обычный Python-файл, размещённый в директории `photon_installer/modules/` и содержащий функцию `run(config, root_mount)`. Во время установки модули автоматически обнаруживаются и выполняются в определённой последовательности.

Пример простого модуля `m_setmotd.py` для добавления кастомного MOTD:

```
def run(config, root_mount):
    motd_path = f"{root_mount}/etc/motd"
    with open(motd_path, "w") as f:
        f.write("Welcome to NiceOS 🚀\\n")
```

Типовые задачи, реализуемые через модули:

- Установка root-пароля (`m_updaterootpassword.py`)
- Добавление SSH-ключей (`m_updatesshconfig.py`)
- Настройка hostname, локали, часового пояса
- Добавление systemd-сервисов, агентов мониторинга
- Регистрация в MDM или инициализация cloud-init

Чтобы добавить собственный модуль, достаточно:

1. Создать Python-файл в `photon_installer/modules/`
2. Определить в нём функцию `run(config, root_mount)`
3. При необходимости — добавить параметр в `ks.json` для управления поведением

Имена модулей должны начинаться с `m_` и иметь уникальное имя, чтобы не конфликтовать с базовыми компонентами.

Использование как Python-библиотеки

Установщик можно использовать не только как CLI-инструмент, но и как встраиваемую библиотеку в Python-скриптах или CI-сценариях. Все ключевые классы

— `Installer`, `IsoInstaller`, `ImageBuilder` — можно импортировать напрямую и вызывать программно.

Пример минимального использования в стороннем скрипте:

```
from photon_installer.installer import Installer

installer = Installer(config_path="ks.json")
installer.configure()
installer.run()
```

Это даёт разработчику полный контроль над поведением установщика: можно вручную вызывать этапы установки, выполнять только разметку, или использовать внутренние утилиты (например, генератор fstab, загрузчик docker-образов и т. д.).

Благодаря такой архитектуре NiceOS Installer может использоваться как часть сборочной системы, автопилота, инструментов управления инфраструктурой или генераторов образов.

Сравнение с популярными установщиками

NiceOS Installer — это легковесный, модульный и ориентированный на автоматизацию установщик, в отличие от традиционных решений, таких как **Anaconda**, **Calamares** или **Debian Installer**. Ниже представлено сравнительное описание по ключевым критериям.

Характеристика	NiceOS Installer	Anaconda	Calamares	Debian Installer
Формат конфигурации	**JSON** (расширяемый, машиночитаемый)	Kickstart (.ks), текстовый	YAML-модули	Preseed (.cfg), текстовый
Графический интерфейс	Нет (CLI / curses / headless)	Да (GTK/TUI)	Да (Qt)	Нет (TUI)
Расширяемость	**Python-модули** (простой API)	Сложная архитектура, требует патчей	C++/Qt, плагины	Ограничена preseed-скриптами
Установка из ISO	Да	Да	Да	Да
Сборка образов (OVA, AMI)	**Да (через isoBuilder)**	Нет	Нет	Нет
Поддержка A/B-обновлений	**Да**	Нет	Нет	Нет

Характеристика	NiceOS Installer	Anaconda	Calamares	Debian Installer
Интеграция с контейнерами / Ansible	**Да**	Ограничено	Нет	Нет
Целевая аудитория	Headless-сборки, CI/CD, серверы, облака	Рабочие станции, корпоративные инсталляции	Настольные дистрибутивы (desktop-focused)	Debian-подобные дистрибутивы, консоль

Когда удобнее применять NiceOS Installer

- **Автоматические сборки ISO/OVA/AMI** в рамках CI/CD
- **Развёртывание headless-серверов** на базе JSON-профилей
- **Массовая установка с USB или по сети** с кастомными параметрами
- **Интеграция с Docker и Ansible** прямо в процессе инсталляции
- **Минимальные зависимости** и высокая предсказуемость в изолированных средах
- **Гибкость** при создании дистрибутивов, использующих A/B-обновления

NiceOS Installer не конкурирует с Anaconda или Calamares в области GUI-установок, а закрывает нишу лёгких, автоматизируемых, скриптуемых и встраиваемых решений для инфраструктурного и DevOps-применения.

Практические примеры

Ниже приведены готовые kickstart-конфигурации, демонстрирующие возможности **NiceOS Installer** в типичных сценариях: быстрое развертывание минимальной системы и создание кастомного образа с дополнительными пакетами и скриптами.

Минимальный kickstart для быстрого развертывания

Подходит для серверов, CI/CD и начальной настройки дистрибутива без лишних компонентов.

```
{
 "hostname": "minimal-node",
 "password": {
  "crypted": false,
  "text": "admin123"
```

```
  },
  "disks": {
    "default": {
      "device": "/dev/sda"
    }
  },
  "partitions": [
    {
      "mountpoint": "/",
      "size": 0,
      "filesystem": "ext4"
    }
  ],
  "packagelist_file": "packages_minimal.json"
}
```

Такой профиль разворачивает минимальную систему с одним разделом и заданным паролем root. Дополнительно можно включить SSH-ключ или Ansible-плейбук.

Создание образа с кастомными пакетами и postinstall-скриптами

Пример конфигурации для генерации ISO-образа с предустановленными пакетами, пользовательским скриптом и Docker-образом:

```
{
  "hostname": "custom-image",
  "password": {
    "text": "securepass"
  },
  "disks": {
    "default": {
      "device": "/dev/vda"
    }
  },
  "partitions": [
    {
      "mountpoint": "/",
      "size": 0,
      "filesystem": "xfs"
    },
    {
      "mountpoint": "/boot",
      "size": 256,
      "filesystem": "ext4"
    }
  ],
```

```
  "packagelist_file": "packages_base.json",
  "additional_packages": [
    "htop",
    "mc",
    "git"
  ],
  "postinstall": [
    "#!/bin/sh",
    "touch /etc/installed-custom",
    "systemctl enable sshd"
  ],
  "niceos_docker_image": "niceos/base:5.0"
}
```

Такой профиль может быть использован в связке с `isoBuilder.py` для генерации установочного ISO:

```
python3 isoBuilder.py --type iso --config ks_custom.json --output niceos-custom.iso
```

Каталог `sample_ks/` содержит дополнительные примеры конфигураций: с LVM, шифрованием, cloud-init, docker-compose и автоматической регистрацией в системе управления.

Заключение

NiceOS Installer — это модульный, расширяемый и легко автоматизируемый установщик, специально созданный для нужд защищённой, инфраструктурной и облачной среды. Он сочетает в себе простоту JSON-конфигураций, гибкость Python-модулей и встроенные средства генерации установочных образов.

Благодаря своей архитектуре он одинаково эффективно работает в роли:

- Установщика с ISO-образа на физические машины и виртуалки
- Инструмента генерации OVA, RAW и AMI-образов
- Компонента в сборочных CI/CD-конвейерах
- Части инфраструктурного решения для массового автозапуска и кастомизации дистрибутива

Планы развития

- Поддержка **network boot (PXE)** с автоматической загрузкой kickstart-конфигураций

- Добавление **GUI-интерфейса (на базе curses или web-UI)** для ручной установки
- Расширение поддержки форматов: **QCOW2, VHDX**, интеграция с облаками
- Поддержка **многоуровневой валидации конфигураций** и шаблонов
- Разработка **SDK для модулей** и библиотек postinstall-расширений
- Улучшенная документация, мастер генерации конфигураций и интерактивные шаблоны

В долгосрочной перспективе NiceOS Installer станет ключевым компонентом экосистемы **NiceOS**, позволяющим не только устанавливать систему, но и формировать предсказуемые, управляемые образы, готовые к эксплуатации в защищённой среде, корпоративной инфраструктуре или облаке.