

# Установка Containerlab

## Linux

Containerlab распространяется как Linux-пакет deb/rpm/apk для архитектур amd64 и arm64 и может быть установлен на любой дистрибутив Debian- или RHEL-подобного типа за несколько секунд.

Для успешной работы Containerlab необходимо заранее:

- Установить Docker или Linux сервер;
- Иметь права sudo для запуска Containerlab;
- Загрузить образы контейнеров (Containerlab попытается загрузить образы во время выполнения, если они не существуют локально).

Самый простой способ начать работу с Containerlab - использовать скрипт быстрой установки quick-setup.sh. Он написан на bash и автоматически определяет тип ОС (Debian, Ubuntu, RHEL, CentOS, Fedora, Rocky), после чего выполняет установку нужных пакетов и зависимостей.

В начале скрипт проверяет:

- что пользователь имеет права sudo (требуется для установки системных пакетов);
- что выполняется на поддерживаемой Linux-системе;
- наличие интернет-соединения для загрузки зависимостей.

Далее при необходимости устанавливается Docker, Containerlab и инструмент GitHub Cli (gh).

Чтобы установить все компоненты сразу, выполните следующую команду на любой из поддерживаемых ОС:

```
curl -sL https://containerlab.dev/setup | sudo -E bash -s "all"
```

По умолчанию это также настроит sshd в системе, чтобы неизвестные ключи не блокировали попытки подключения по SSH. Это поведение можно отключить, установив переменную окружения SETUP\_SSHD в значение false перед запуском приведённой выше команды. Переменная окружения устанавливается и экспортируется следующей командой:

```
export SETUP_SSHD="false"
```

Чтобы завершить установку и включить выполнение команд `docker` без `sudo`, выполните `newgrp docker` или выйдите из системы и войдите снова.

Containerlab также настраивается для работы без `sudo`, и пользователь, выполняющий скрипт быстрой установки, автоматически получает доступ к привилегированным командам.

Чтобы установить отдельный компонент, укажите имя функции в качестве аргумента скрипта. Например, чтобы установить только `docker`:

```
curl -sL https://containerlab.dev/setup | sudo -E bash -s "install-docker"
```

Выйдите из текущей сессии пользователя и войдите снова, чтобы увидеть новое двухстрочное приглашение в действии:

```
[*]—[clab]—[~]  
└─>
```

Если вам не надо устанавливать дополнительные инструменты, а необходим лишь один Containerlab можно воспользоваться скриптом установки. Например, чтобы скачать и установить последнюю версию:

```
bash -c "$(curl -sL https://get.containerlab.dev)"
```

Также можно установить официальные выпуски containerlab через публичный репозиторий APT/YUM.

## APT

```
echo "deb [trusted=yes] https://netdevops.fury.site/apt/ /" | \  
sudo tee -a /etc/apt/sources.list.d/netdevops.list  
  
sudo apt update && sudo apt install containerlab
```

## YUM

```
sudo yum-config-manager --add-repo=https://netdevops.fury.site/yum/ && \  
(https://containerlab.dev/install/#__codeline-9-2)echo "gpgcheck=0" | sudo  
tee -a /etc/yum.repos.d/netdevops.fury.site_yum_.repo [\  
(https://containerlab.dev/install/#__codeline-9-3)[\  
(https://containerlab.dev/install/#__codeline-9-4)sudo yum install  
containerlab
```

Пакетный установщик поместит бинарный файл containerlab в каталог `/usr/bin`, а также создаст символическую ссылку `/usr/bin/clab -> /usr/bin/containerlab`. Эта ссылка позволяет пользователям экономить на наборе команд при использовании containerlab: `clab`. Containerlab также настраивается для работы без `sudo`, и текущий пользователь (даже если менеджер пакетов был вызван через `sudo`) автоматически получает доступ к привилегированным командам Containerlab.

## Установка с помощью контейнера

Containerlab также доступен в виде образа контейнера. Последнюю версию можно загрузить командой:

```
docker pull ghcr.io/srl-labs/clab
```

Чтобы выбрать любую из выпущенных версий начиная с **0.19.0**, используйте номер версии в качестве тега, например:

```
docker pull ghcr.io/srl-labs/clab:0.19.0
```

Загруженный контейнер запускается командой:

```
docker run --rm -it --privileged \
  --network host \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /var/run/netns:/var/run/netns \
  -v /etc/hosts:/etc/hosts \
  -v /var/lib/docker/containers:/var/lib/docker/containers \
  --pid="host" \
  -v $(pwd):$(pwd) \
  -w $(pwd) \
  ghcr.io/srl-labs/clab bash
```

Рассмотрим основные требуемые привилегии:

- `--privileged`: полный доступ к хосту для создания и управления сетевым пространством;
- `--network host`: доступ к сетевому стеку хоста;
- `--pid="host"`: доступ к пространству процессов хоста.

В запущенном контейнере можно использовать те же команды **deploy/destroy/inspect** для управления лабораториями.

Можно развернуть лабораторию альтернативным способом без запуска оболочки, например:

```
docker run --rm -it --privileged \  
# <параметры опущены>  
-w $(pwd) \  
ghcr.io/srl-labs/clab deploy -t somelab.clab.yml
```

## Ручная установка

Если дистрибутив Linux не поддерживает установку deb/rpm пакетов, containerlab можно установить из архива:

```
LATEST=$(curl -s https://github.com/srl-labs/containerlab/releases/latest | \  
 \  
    sed -e 's/.*tag\v\(.*\)\\".*\1/')  
curl -L -o /tmp/clab.tar.gz "https://github.com/srl-  
labs/containerlab/releases/download/v${LATEST}/containerlab_${LATEST}_Linux_  
amd64.tar.gz"  
mkdir -p /etc/containerlab  
tar -zxvf /tmp/clab.tar.gz -C /etc/containerlab  
mv /etc/containerlab/containerlab /usr/bin && chmod a+x  
/usr/bin/containerlab
```

## Обновление

Чтобы обновить containerlab до последней доступной версии, выполните:

```
sudo -E containerlab version upgrade
```

Эта команда скачает скрипт установки и обновит инструмент до последней версии.

## С помощью исходников

Чтобы собрать containerlab из исходников есть два варианта:

- **go build**;
- **goreleaser**.

В первом случае, требуется клонировать репозиторий, перейти в корневой каталог и выполнить команду `go build`:

```
git clone https://github.com/srl-labs/containerlab
cd containerlab
go build
```

Во втором случае, для сборки запускается:

```
goreleaser --snapshot --skip-publish --rm-dist
```

## Удаление

Отдельно рассмотрим удаление инструмента.

Чтобы удалить containerlab, установленный через скрипт или пакеты:

- **Для Debian-систем:**

```
apt remove containerlab
```

- **Для RPM-систем:**

```
yum remove containerlab
# or
dnf remove containerlab
```

## Windows

Для того, чтобы развернуть Containerlab на Windows, требуется использовать WSL ([Windows Subsystem Linux](#)). WSL позволяет пользователям запускать легковесные Linux VM внутри Windows, это можно использовать для развертывания Containerlab.

Если WSL не установлен, исправить это можно одной командой:

```
wsl --install
```

Далее рассмотрим установку Containerlab. В Powershell от имени администратора включаем компоненты Windows, необходимые для работы подсистемы Linux: активируем базовую функциональность WSL и платформу виртуальных машин для использования более производительной версии WSL 2:

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

После выполнения этих команд и перезагрузки компьютера, нужно с помощью команды `wsl --set-default-version 2` задать WSL 2 как версию по умолчанию.

Загрузим файл `.wsl` со [страницы релизов](#). Просто дважды щелкнем по файлу, и дистрибутив будет установлен. Альтернативно его можно установить с помощью `wsl --install -from-file C:\path\to\clab.wsl`. Containerlab должен появиться как программа в меню "Пуск".

Запустим Containerlab:

```
wsl -d Containerlab
```

При первом запуске Containerlab WSL выполняется начальный этап настройки, который помогает настроить дистрибутив. Настраивается оболочка и командная строка. После этого SSH-ключи будут скопированы или сгенерированы (если ключей не существует). Это необходимо для обеспечения доступа к SSH без пароля, что улучшает интеграцию с DevPod.

## macOS

Запуск containerlab на macOS возможен как на ARM (M1/M2/M3 и т.д.), так и на Intel-чипах. Долгое время существовали ограничения для ARM-чипов, но с появлением ARM64-совместимых сетевых ОС, таких как Nokia SR Linux и Arista cEOS, а также благодаря эмуляции Rosetta для x86\_64-систем, теперь можно запускать containerlab и на Mac с ARM чипом.

Для работы Containerlab на macOS используется Linux-виртуальная среда, поскольку macOS не имеет нативной поддержки Linux-сетевых пространств. В качестве такой среды рекомендуется использовать OrbStack — лёгкий гипервизор, совместимый с Docker и Linux-ядром.

## Порядок установки

1. **Установить OrbStack.** Скачать и установить OrbStack с официального сайта <https://orbstack.dev>. OrbStack создаёт лёгкую виртуальную машину Linux, в которой работает Docker и все сетевые инструменты, необходимые Containerlab.
2. **Запустить Linux-среду внутри OrbStack.** После установки открыть терминал OrbStack и войти в Linux-оболочку (по умолчанию Ubuntu-подобная система).

3. **Установить Docker и Containerlab.** Внутри OrbStack-Linux выполнить стандартную установку для Linux: `curl -sL https://containerlab.dev/setup | sudo -E bash -s "all"`. Эта команда установит Docker, Docker Compose, GitHub CLI и сам Containerlab.

```
nasmi@ubuntu:~$ curl -sL https://containerlab.dev/setup | sudo -E bash -s "all"
sed: can't read /etc/ssh/sshd_config: No such file or directory
Failed to restart ssh.service: Unit ssh.service not found.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package 'docker.io' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 9 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package 'docker-doc' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 9 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package 'docker-compose' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 9 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package 'docker-compose-v2' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 9 not upgraded.
Reading package lists... Done
Building dependency tree... Done
```

4. **Проверить установку.** После завершения выйти и войти снова, чтобы активировать группы пользователей, и проверить:

```
clab version
docker ps
```

```
nasmi@ubuntu:~$ clab version
```



```
version: 0.71.0
commit: 7ef796f07
date: 2025-10-10T17:36:13Z
source: https://github.com/srl-labs/containerlab
rel. notes: https://containerlab.dev/rn/0.71/
```

```
nasmi@ubuntu:~$ docker ps
```

```
permission denied while trying to connect to the Docker daemon socket at unix
2Frun%2Fdocker.sock/v1.47/containers/json": dial unix /var/run/docker.sock: c
```

```
nasmi@ubuntu:~$ newgrp docker
```

```
nasmi@ubuntu:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

## 5. Проверить архитектуру сетевых образов. На ARM-Мас рекомендуется

использовать контейнерные образы, собранные под архитектуру arm64. Проверить  
это можно командой:

```
docker image inspect <image> -f '{{.Architecture}}'
```

## 6. Развернуть лабораторию. Теперь можно развернуть тестовую лабораторию,

например: `sudo clab deploy -t examples/srl01.clab.yml`.

```
nasmi@ubuntu:~/work/containerlab/lab-examples/srl-quickstart$ clab deploy -t srl02.clab.yml
18:25:22 INFO Containerlab started version=0.71.0
18:25:22 INFO Parsing & checking topology file=srl02.clab.yml
18:25:22 INFO Creating docker network name=clab IPv4 subnet=172.20.20.0/24 IPv6 subnet=3fff:172:20:20::/64 MTU=0
18:25:22 INFO Creating lab directory path=/home/nasmi/work/containerlab/lab-examples/srl-quickstart/clab-srl2
18:25:22 INFO Creating container name=srl2
18:25:22 INFO Creating container name=srl1
18:25:22 INFO Running postdeploy actions kind=nokia_srlinux node=srl1
18:25:22 INFO Created link: srl1:e1-1 ──── srl2:e1-1
18:25:22 INFO Created link: srl1:e1-2 ──── srl2:e1-2
18:25:22 INFO Running postdeploy actions kind=nokia_srlinux node=srl2
18:25:35 INFO Adding host entries path=/etc/hosts
18:25:35 INFO Adding SSH config for nodes path=/etc/ssh/ssh_config.d/clab-srl2.conf
You are on the latest version (0.71.0)
```

Name	Kind/Image	State	IPv4/6 Address
srl1	nokia_srlinux ghcr.io/nokia/srlinux:latest	running	172.20.20.2 3fff:172:20:20::2
srl2	nokia_srlinux ghcr.io/nokia/srlinux:latest	running	172.20.20.3 3fff:172:20:20::3

```
nasmi@ubuntu:~/work/containerlab/lab-examples/srl-quickstart$
```

Зайдем на первый хост, посмотрим существующие интерфейсы и поднимем интерфейс  
ethernet-1/1, добавим ipv4 адрес 10.0.0.1/31.



```

nasm@ubuntu:~/work/containerlab/lab-examples/srl-quickstart$ docker exec -it srl1 sr_cli
Loading environment configuration file(s): []
Welcome to the Nokia SR Linux CLI.

--{ + running }--[ ]--
A:root@srl1# show interface
=====
ethernet-1/1 is up, speed 100G, type None
  ethernet-1/1.0 is down, reason no-ip-config
  Network-instances:
    Encapsulation   : null
    Type            : routed
    IPv4 addr       : 10.0.0.1/31 (static, None)
-----

ethernet-1/2 is up, speed 100G, type None
-----

mgmt0 is up, speed 1G, type None
  mgmt0.0 is up
  Network-instances:
    * Name: mgmt (ip-vrf)
    Encapsulation   : null
    Type            : None
    IPv4 addr       : 172.20.20.2/24 (dhcp, preferred)
    IPv6 addr       : 3fff:172:20:20::2/64 (dhcp, preferred)
    IPv6 addr       : fe80::42:acff:fe14:1402/64 (link-layer, preferred)
-----

Summary
  0 loopback interfaces configured
  2 ethernet interfaces are up
  1 management interfaces are up
  1 subinterfaces are up
=====

--{ + running }--[ ]--
A:root@srl1# enter candidate

--{ + candidate shared default }--[ ]--
A:root@srl1# interface ethernet-1/1

--{ + candidate shared default }--[ interface ethernet-1/1 ]--
A:root@srl1# admin-state enable

--{ +* candidate shared default }--[ interface ethernet-1/1 ]--
A:root@srl1# subinterface 0

--{ +* candidate shared default }--[ interface ethernet-1/1 subinterface 0 ]--
A:root@srl1# ipv4 address 10.0.0.1/31

--{ +* candidate shared default }--[ interface ethernet-1/1 subinterface 0 ipv4 address 10.0.0.1/31 ]--
A:root@srl1# commit now

```

Проверка добавления интерфейса.

```
--{ + running }--[ ]--
A:root@srl1# /show interface ethernet-1/1
=====
ethernet-1/1 is up, speed 100G, type None
  ethernet-1/1.0 is down, reason no-ip-config
    Network-instances:
      Encapsulation    : null
      Type              : routed
      IPv4 addr        : 10.0.0.1/31 (static, None)
-----
=====

--{ + running }--[ ]--
A:root@srl1#
```

Аналогично делаем на втором узле. Присваиваем ipv4 адрес 10.0.0.2/31.

```
nasmi@ubuntu:~/work/containerlab/lab-examples/srl-quickstart$ docker exec -it srl2 sr_cli
Loading environment configuration file(s): []
Welcome to the Nokia SR Linux CLI.

--{ running }--[ ]--
A:root@srl2# show interface
=====
ethernet-1/1 is up, speed 100G, type None
-----
ethernet-1/2 is up, speed 100G, type None
-----
mgmt0 is up, speed 1G, type None
  mgmt0.0 is up
    Network-instances:
      * Name: mgmt (ip-vrf)
    Encapsulation   : null
    Type            : None
    IPv4 addr       : 172.20.20.3/24 (dhcp, preferred)
    IPv6 addr       : 3fff:172:20:20::3/64 (dhcp, preferred)
    IPv6 addr       : fe80::42:acff:fe14:1403/64 (link-layer, preferred)
    -----
Summary
  0 loopback interfaces configured
  2 ethernet interfaces are up
  1 management interfaces are up
  1 subinterfaces are up
  -----

--{ running }--[ ]--
A:root@srl2# enter candidate
interface ethernet-1/1
  admin-state enable
  subinterface 0
    admin-state enable
    ipv4 address 10.0.0.2/31
commit now
All changes have been committed. Leaving candidate mode.

--{ + running }--[ interface ethernet-1/1 subinterface 0 ipv4 address 10.0.0.2/31 ]--
A:root@srl2#
```

```
A:root@srl2# /show interface
```

```
=====
ethernet-1/1 is up, speed 100G, type None
  ethernet-1/1.0 is down, reason no-ip-config
  Network-instances:
    Encapsulation   : null
    Type            : routed
    IPv4 addr       : 10.0.0.2/31 (static, None)
  =====
```

```
ethernet-1/2 is up, speed 100G, type None
=====
```

```
mgmt0 is up, speed 1G, type None
  mgmt0.0 is up
  Network-instances:
    * Name: mgmt (ip-vrf)
    Encapsulation   : null
    Type            : None
    IPv4 addr       : 172.20.20.3/24 (dhcp, preferred)
    IPv6 addr       : 3fff:172:20:20::3/64 (dhcp, preferred)
    IPv6 addr       : fe80::42:acff:fe14:1403/64 (link-layer, preferred)
  =====
```

#### Summary

```
0 loopback interfaces configured
2 ethernet interfaces are up
1 management interfaces are up
1 subinterfaces are up
=====
```

```
--{ + running }--[ interface ethernet-1/1 subinterface 0 ipv4 address 10.0.0.2/31 ]--
A:root@srl2#
```

Пропингуем первый хост со второго: ping 10.0.0.1

С помощью команды `containerlab graph -t srl02.clab.yml` посмотрим на графическое изображение сети.

## ContainerLab Topology [SRL2](#)

Horizontal Layout

Vertical Layout

