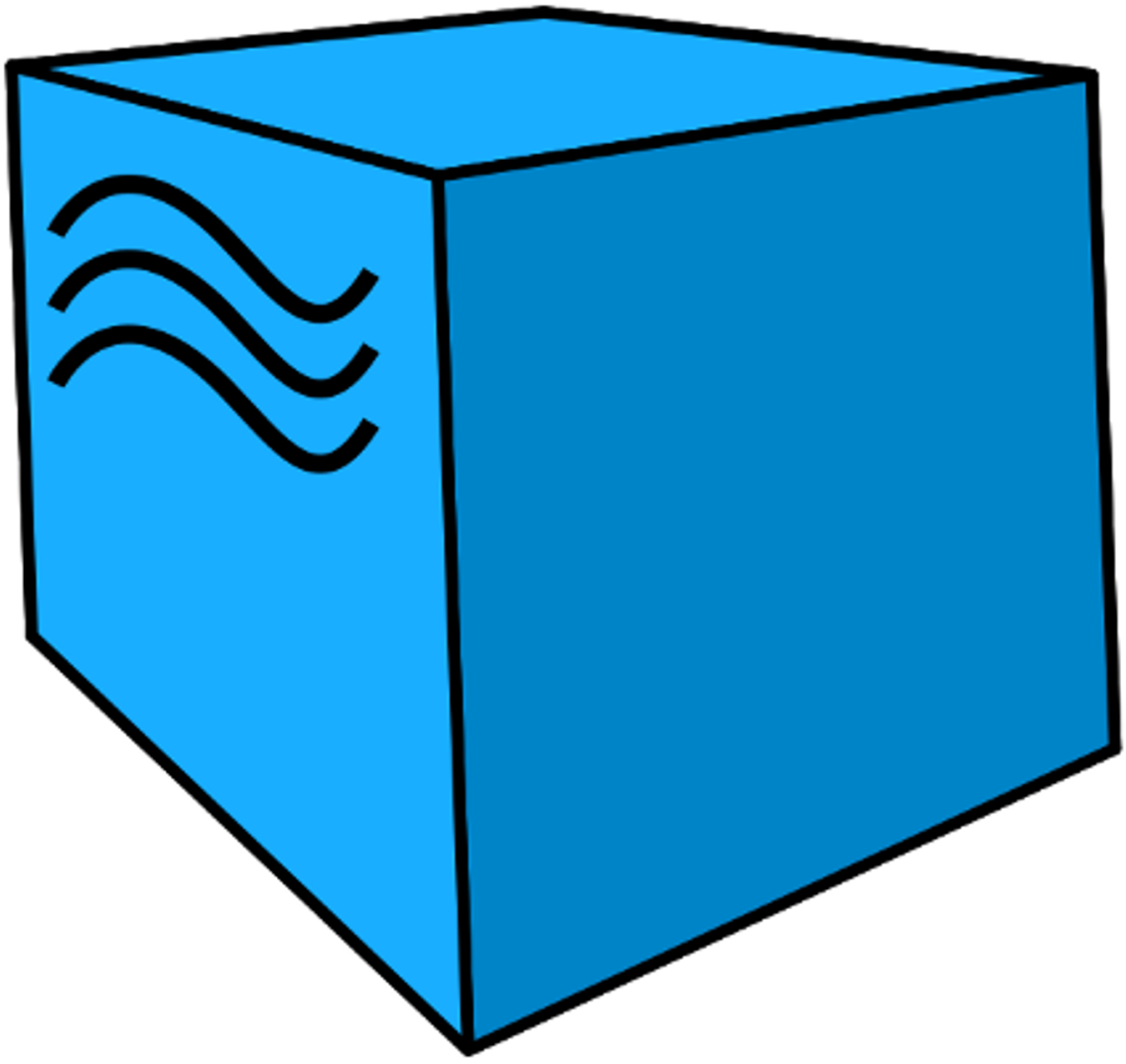
Aerokube Boot



https://aerokube.ru/

Общее Руководство по ПО "Boot"

Содержание

1. О документе

Документ предназначен для технических специалистов, которые установили и занимаются эксплуатацией ПО "Boot".

2. Начало работы

2.1. Быстрый старт

Этот раздел описывает установку Boot с ограничением в 1 процессор. Подробную информацию о том как установить лицензионный ключ, позволяющий использовать больше процессоров можно получить в разделе Лицензионный ключ.

2.1.1. Установка в Kubernetes

Системные требования

1. Работающий кластер Kubernetes.
2. Установленный `kubectl` клиент с настроенным доступом к кластеру
3. Пара ключей ssh, которые обычно хранятся в директории ~/.ssh/ под именами `id_rsa` (приватный ключ) и `id_rsa.pub` (публичный ключ). Если ключей нет, необходимо их сгенерировать командой `ssh-keygen`. Для установки Boot потребуется только публичный ключ, который обычно выглядит так:

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3N... some-comment
```

4. При установке Boot в кластер Kubernetes, запущенный на рабочей станции, при помощи инструмента minikube - кластер Kubernetes нужно запускать одной из следующих команд:

Запуск Minikube на Linux

```
$ minikube start --driver kvm2
```

Запуск Minikube на MacOS

```
$ minikube start --driver=hyperkit
```

Запуск Minikube на Windows

```
$ DISM /Online /Enable-Feature /All /FeatureName:Microsoft-Hyper-V # Enable Hyper-V
$ minikube start --driver=hyperv
```

Установка Boot при помощи Helm

ВАЖНО: Helm чарты (пакеты) являются рекомендуемым инструментом установки Boot. Дальнейшие шаги предполагают установленный Helm 3. Более старые версии не поддерживаются.

Мы предоставляем готовые Helm чарты, что позволяет установить Boot с помощью Helm простой при помощи нескольких команд:

1. Создайте файл `values.yaml` с конфигурацией Helm чарта:

```
boot:
  jumphost:
    authorizedKeys:
      - ssh-rsa AAAAB3N... some-comment # Вставьте содержимое хотя бы одного
публичного ключа в данный раздел
```

2. Добавьте репозиторий с Helm чартами Aerokube charts:

```
$ helm repo add aerokube https://charts.aerokube.ru/
$ helm repo update
```

3. Для вывода списка доступных версий Boot используйте команду:

```
$ helm search repo aerokube --versions
```

4. Создайте Kubernetes неймспейс:

```
$ kubectl create namespace boot
```

5. Установите или обновите Boot командой:

```
$ helm upgrade --install -f values.yaml -n boot boot aerokube/boot
```

6. Helm чарт для Boot содержит различные параметры, которые можно посмотреть командой:

   ```
   $ helm show values aerokube/boot
   ```

   Для изменения одного из этих параметров - переопределите его в файле `values.yaml` и выполните команду установки Helm чарта еще раз.

7. Выясните IP адрес балансировщика нагрузки Boot. Как правило, адрес балансировщика может быть получен из сервиса Boot:

   Как узнать IP адрес сервиса Boot

   ```
   $ kubectl get svc boot -n boot
   NAME   TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)    AGE
   boot   LoadBalancer   10.107.117.163   192.168.64.5   2222/TCP   15d
   ```

   IP адрес указан в колонке `EXTERNAL-IP`. При использовании Minikube IP адрес необходимо добавить вручную, с помощью вывода команды `minikube ip`:

   Добавление IP адреса в сервис Boot вручную

   ```
   $ kubectl patch svc boot -n boot --patch "{\"spec\":{\"externalIPs\":[\"$(minikube
   ip)\"]}}"
   ```

   На Windows вывод команды `minikube ip` необходимо подставить вручную, поскольку выражение `$()` может не сработать.

8. Удостоверьтесь, что доменное имя указывает на этот IP адрес. При использовании Minikube - просто отредактируйте файл `hosts`:

   ```
   $ sudo echo "$(minikube ip) boot.aerokube.local" >> /etc/hosts
   ```

   На Windows вам необходимо отредактировать файл `hosts` вручную.

9. Сконфигурируйте SSH клиент для использования Boot. Для этого добавьте следующие параметры в файл `~/.ssh/config`:

   ```
   $ cat ~/.ssh/config
   # ... другие записи
   Host bootjump
       IdentityFile ~/.ssh/id_rsa
       HostName boot.aerokube.local
       Port 2222
   ```

```
Host *.vm.boot.svc.cluster.local
    IdentityFile ~/.ssh/id_rsa
    ProxyJump jump@bootjump
```

10. Создайте вашу первую виртуальную машину. Для этого создайте YAML файл с описанием структуры виртуальной машины:

```
$ cat ~/vm.yaml
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: boot
spec:
  os: ubuntu
  authorizedKeys:
    - ssh-rsa AAAAB3N... some-comment # The same public key contents here too
```

Создайте виртуальную машину используя этот файл:

```
$ kubectl create -f vm.yaml
```

11. Проверьте доступность виртуальной машины по SSH:

```
$ ssh root@my-vm.vm.boot.svc.cluster.local
# Здесь идет некоторый приветственный текст...
root@my-vm:~# # Теперь у вас есть root доступ к виртуальное машине
```

2.2. Архитектура

2.2.1. Компоненты Boot

Компоненты Boot

[boot components]

Кластер Boot состоит из нескольких компонентов:

1. Одна или несколько реплик `Boot`. Они запускают/останавливают виртуальные машины и следят за доступом по SSH к созданным машинам. `Boot` обычно доступен по какому либо из SSH портов, например `2222`.
2. Одна или несколько реплик `Boot UI` (эта фунциональность в данный момент ещё находится в разработке). `Boot UI` собирает информацию от Boot и визуализирует её. Обычно интерфейс доступен на HTTP порту `8080`.
3. Объектов виртуальных машин и соответствующих им запущенных подов.

2.2.2. Содержание пода Boot

Каждый под Boot содержит несколько контейнеров для выполнения специфичных задач.

Table 1. Контейнеры внутри пода Boot

Имя	Назначение
boot	Стартует и останавливает виртуальные машины
jumphost	Стандартный SSH сервер с защищенной конфигурацией работающий как SSH шлюз (SSH сервер в режиме jump host)
reloader	Во время обновления Boot в течение заданного периода следит за SSH соединениями к виртуальным машинам, чтобы закрыть или переоткрыть их после завершения обновления

2.2.3. Поддержка нескольких неймспейсов

Boot может запускать виртуальные машины на любом количестве неймспейсов в Kubernetes. По умолчанию сам Boot и все запущенные виртуальные машины работают в одном неймспейсе. В этом нет проблемы, если Boot используется только одной командой либо если нет необходимости ограничивать доступные вычистительные ресурсы виртуальных машин для разных пользователей Boot.

Boot настроенный для работы с несколькими неймспейсами

[multiple-namespaces-mode]

Можно настроить Boot таким образом, чтобы он был запущен в одном неймспейсе и управлял виртуальными машинами в одном или нескольких соседних неймспейсах. Необходимость в нескольких неймспейсах обычно имеется тогда, когда нужно контролировать/ограничивать вычислительные ресурсы, запускать виртуальные машины с разными релизами операционных систем либо вследствие особенностей настроек сетевого доступа (network policies) для каждой команды. Например, виртуальные машины одного пользователя будут работать в неймспейсе одной команды сотрудников, а виртуальные машины второго пользователя будут работать в неймспейсе второй команды. Информацию о том как сконфигурировать несколько неймспейсов можно получить по ссылке.

2.3. Необходимые права доступа

Boot не требует широких прав доступа в Kubernetes. Обычно ему достаточно настроек доступа Kubernetes по умолчанию. Boot может запускать виртуальные машины на любом количестве неймспейсов Kubernetes. Все необходимые права и роли автоматически создаются Helm чартом. Следующая таблица показывает, какие права доступа Boot имеет в каждом в неймспейсе:

Table 2. Необходимые права доступа

Роль	**Назначение**
Чтение (**get**), подписка (**watch**), вывод списка (**list**) и редактирование (**patch**) ресурсов самого Boot в группе `boot.aerokube.com`	Эти ресурсы хранят конфигурацию Boot и доступны во всем кластере, поэтому требуется добавлять ClusterRole (роль на весь кластер Kubernetes).

Необходимые права для пользователя:

Table 3. Права пользователя

Роль	**Назначение**
Чтение (**get**), подписка (**watch**), вывод списка (**list**), создание (**create**), удаление (**delete**), обновление (**update**) и редактирование (**patch**) подов	Используется для управления виртуальными машинами
Чтение (**get**), подписка (**watch**), вывод списка (**list**), создание(**create**), удаление (**delete**), обновление (**update**) и редактирование (**patch**) config maps	Используется для передачи списка пользователей и групп на виртуальные машины

3. Основные возможности

3.1. Виртуальные Машины

Boot позволяет создавать легковесные виртуальные машины. Эти виртуальные машины могут быть использованы для:

- Функционального тестирования, то есть проверки того, что программное обеспечение работает так, как это было задумано.
- Тестирования безопасности, то есть проверки того, что программное обеспечение не содержит уязвимостей.
- Тестирование производительности, то есть проверка поведения программного обеспечения в условиях определенной нагрузки.
- Разработка программного обеспечения. Виртуальные машины дают больше вычислительных ресурсов (количество процессоров, памяти, дискового пространства), чем обычный рабочий компьютер разработчика.

Виртуальные машины Boot, как и обычные виртуальные машины, доступны по SSH. Это означает что вы можете использовать любые стандартные инструменты для обновления состояния машины:

- Infrastructure as code инструменты (Terraform, Ansible и так далее) для установки дополнительного ПО на машину.
- IDE для работы с кодом на виртуальной машине удаленно. Самые известные

инструменты, реализующие данную функциональность это Visual Studio Code и Jetbrains IDEs. Их использование совместно с Boot описано в следующих разделах данной документации.

3.1.1. Вывод списка виртуальных машин

Виртуальные машины Boot являются встроенными объектами Kubernetes, поэтому вывод списка машин делается стандартной командой `kubectl`:

Вывод списка виртуальных машин

```
$ kubectl get vm -n boot
NAME    FQDN                                IP           OS            CPU         MEM
STATUS    UPTIME
my-vm   my-vm.vm.boot.svc.cluster.local   172.17.0.4   alpine:3.18   250m/128m   512Mi
Running   4s
```

В списке видно имя виртуальной машины, доменное имя, IP адрес, использованный образ, информация о числе доступных вычислительных ядер, количестве оперативной памяти, состояние и время работы.

3.1.2. Создание виртуальных машин

Способ 1. Используя kubectl и YAML манифест

1. Создайте YAML файл, описывающий виртуальную машину:

   ```
   $ cat vm.yaml
   apiVersion: boot.aerokube.com/v1
   kind: VirtualMachine
   metadata:
     name: my-vm
     namespace: boot
   spec:
     os: ubuntu
     authorizedKeys: # Добавьте один или несколько публичных SSH ключей
       - ssh-rsa AAAAB3N... some-key
       - ssh-rsa AAAAB3N... another-key
   ```

2. Создайте виртуальную машину используя этот файл:

   ```
   $ kubectl create -f ~/vm.yaml
   virtualmachine.boot.aerokube.com/my-vm created
   ```

   После этого вы можете вывести список виртуальных машин и получить полную информацию о доменном имени, виртуальная машина доступна и готова к работе. Как зайти на виртуальную машину описано в разделе Доступ к виртуальным машинам.

Способ 2. Используя Boot плагин для kubectl

Второй способ создания виртуальных машин Boot - используя плагин для kubectl.

Самый простой способ установки Boot плагин - через пакетный менеджер Krew:

Установка Boot плагина с помощью Krew

```
$ kubectl krew index add aerokube https://github.com/aerokube/krew-index.git
$ kubectl krew install aerokube/vm
```

Другой способ - загрузка архивированного плагина и распаковка в директорию PATH:

Установка Boot плагина вручную на Mac

```
$ curl -o kubectl-vm.zip https://download.aerokube.com/boot/kubectl-vm/latest-
release/kubectl-vm_darwin_amd64.zip # x86 version
$ curl -o kubectl-vm.zip https://download.aerokube.com/boot/kubectl-vm/latest-
release/kubectl-vm_darwin_arm64.zip # ARM version
$ unzip -d /usr/local/bin kubectl-vm.zip
```

Установка Boot плагина вручную на Linux

```
$ curl -o kubectl-vm.zip https://download.aerokube.com/boot/kubectl-vm/latest-
release/kubectl-linux_amd64.zip
$ unzip -d /usr/local/bin kubectl-vm.zip
```

Установка Boot плагина вручную на Windows

```
> curl -o kubectl-vm.zip https://download.aerokube.com/boot/kubectl-vm/latest-
release/kubectl-windows_amd64.zip
> mkdir %USERPROFILE%\bin
> Expand-Archive -Force kubectl-vm.zip %USERPROFILE%\bin # Добавьте %USERPROFILE%\bin
в переменную PATH
```

Проверка правильности установки плагина:

```
$ kubectl vm --help
```

Создание виртуальной машины с помощью плагина:

Использование плагина

```
$ kubectl vm create my-vm -os ubuntu -n boot # Используем последнюю доступную версию
операционной системы
$ kubectl vm create my-vm -os ubuntu:22.04 -n boot # Явное указание версии
операционной системы
```

По-умолчанию, плагин использует публичный ключ из файла `~/.ssh/id_rsa.pub`, но также можно явно передать список доверенных публичных ключей:

Явная передача списка доверенных публичных ключей

```
$ echo "ssh-rsa AAAAB3N... some-key" >> authorized_keys # Создаем файл со списком
публичных ключей
$ kubectl vm create my-vm -os ubuntu -auth-keys authorized_keys -n boot # Используем
полученный файл при создании виртуальной машины
```

3.1.3. Удаление виртуальных машин

Для удаления виртуальной машины используйте стандартную команду `kubectl`:

Удаление с помощью kubectl

```
$ kubectl delete vm my-vm -n boot
```

Удаление с помощью плагина для `kubectl`:

Удаление виртуальной машины с помощью плагина kubectl

```
$ kubectl vm delete my-vm -n boot
```

3.1.4. Контроль доступа

Boot поддерживает две основные технологии SSH аутентификации: **указание доверенных ключей** и **использование сертификатов**. Принципы работы и различия описаны в разделе Доступ к виртуальным машинам. Для того чтобы добавить доверенные публичные ключи, используйте поле `authorizedKeys` в описании виртуальной машины:

Добавление ключей

```
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: boot
spec:
  os: ubuntu
  authorizedKeys: # Список публичных ключей, владельцем которых разрешено заходить на
виртуальную машину
    - ssh-rsa AAAAB3N... some-key
    - ssh-rsa AAAAB3N... another-key
```

Для настройки доступа пользователей с помощью SSH сертификатов необходимо добавить сертификаты удостоверяющих центров (certification authorities, CA), которыми подписаны пользовательские сертификаты. Для этого в описании виртуальной машины используется

поле `trustedUserCAKeys`:

Добавление сертификатов удостоверяющих центров

```
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: boot
spec:
  os: ubuntu
  trustedUserCAKeys: # Список доверенных сертификатов удостоверяющих центров
    - ssh-ed25519 AAAAC3N... certification-authority-1
    - ssh-ed25519 AAAAC3N... certification-authority-2
```

Разница между ключами и сертификатами описывается здесь.

3.1.5. Настройка вычислительных ресурсов

Boot использует тот же формат описания вычислительных ресурсов в формате YAML, что и Kubernetes.

По умолчанию виртуальная машина использует вычислительные ресурсы сконфигурированные в #operating-systems-computing-resources[настройках операционной системы]. Изменить конфигурацию по умолчанию можно в описании виртуальной машины:

Настройка вычислительных ресурсов

```
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: boot
spec:
  os: ubuntu
  resources: # Явно задание вычислительных ресурсов
    limits:
      cpu: 1000m
      memory: 512Mi
    requests:
      cpu: 1000m
      memory: 512Mi
```

Также настройки могут быть заданы и при создании виртуальной машины с помощью плагина kubectl:

Настройка вычислительных ресурсов при помощи плагина

```
$ kubectl vm create -os ubuntu -cpu 1.0 -mem 512Mi -auth-keys authorized_keys -n boot
$ kubectl vm create -os ubuntu -cpu '1.0/0.5' -mem '512Mi/256Mi' -auth-keys
authorized_keys -n boot # Разные значения для requests и limits
```

3.1.6. Подключение хранилищ данных (volumes)

По умолчанию Boot использует ту же конфигурацию хранилища данных в YAML формате, что и Kubernetes.

Общий доступ к файлам с нескольких виртуальных машин, либо возможность сохранения файла осуществляется с помощью механизма подключаемых хранилищ данных (volumes). Можно добавить любой тип подключаемого хранилища, поддерживаемый вашим Kubernetes кластером. Для подключения хранилища:

Подключение хранилища

```
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: boot
spec:
  os: ubuntu
  volumes: # Мы объявляем хранилище
  - name: my-volume
    emptyDir: {}
  volumeMounts: # Мы подключаем его в определенный каталог виртуальной машины
  - name: my-volume
    mountPath: /some/path
```

3.1.7. Изменение содержимого файла hosts

В некоторых случаях возникает необходимость изменить/добавить информацию в файл `/etc/hosts` внутри виртуальной машины. Решение этой задачи выглядит так:

Обновление /etc/hosts

```
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: boot
spec:
  os: ubuntu
  hostAliases:
    - ip: "127.0.0.1"
      hostnames:
```

```
      - "foo.local"
      - "bar.local"
    - ip: "10.1.2.3"
      hostnames:
      - "foo.remote"
      - "bar.remote"
```

3.2. Доступ к виртуальным машинам

Мы уже упоминали в разделе Виртуальные Машины, что виртуальные машины Boot доступны по протоколу SSH. Этот раздел описывает возможности настройки безопасного доступа к виртуальным машинам.

3.2.1. Обзор

Протокол SSH состоит из двух компонентов: SSH сервер (также известный как `sshd`) и SSH клиент (обычно называемый просто `ssh`). SSH сервер в большинстве случаев ожидает соединения по стандартному TCP протоколу на порту 22. Boot запускается в Kubernetes и таким образом любой сетевой порт, к которому вам нужно иметь доступ извне, должен быть настроен как сервис Kubernetes. Наша цель - иметь возможность запускать неограниченное количество виртуальных машин, поэтому создание отдельного сервиса для каждой из них было бы слишком трудоемким. Кроме того мы хотим иметь возможность защитить все созданные виртуальные машины от несанкционированного доступа. SSH дает возможность достичь обеих целей стандартным способом - используя Jump host. Jump host это специальный хост, к которому подключается SSH клиент для последующего доступа к нужной виртуальной машине.

Jump host

[jump host]

Из соображений безопасности Boot по умолчанию использует нестандартный порт `2222` для доступа к jump host. Причина этого заключается в том, что в операционных системах семейства Unix порты до 1024 являются привилегированными и требуют прав суперпользователя (`root`) для использования. Кроме этого в Kubernetes рекомендуется использовать непривилигерованные порты там, где это возможно. Когда вы устанавливаете Boot с настройками по умолчанию - jump host настраивается автоматически и запускается сразу после завершения установки. Описание конфигурации jump host по-умолчанию, а так же как изменить настройки по-умолчанию доступно в разделе jump host.

3.2.2. Аутентификация

SSH поддерживает два основных метода аутентификации: **ключи** (более простой и более популярный способо) и **сертификаты** (более сложный, но более надежный способ). Оба метода основаны на шифровании с открытым ключом. Шифрование с открытым ключом в свою очередь основывается на использовании односторонних математических функций. Такие функции легко вычисляются для любого входного значения, но, даже с помощью современных компьютеров трудно найти аргумент по заданному значению функции. К

примеру, задача поиска простых делителей лежит в основе широко используемого алгоритма RSA. Алгоритмы ECDSA (использующийся, в частности, для проверки электронной подписи) и EdDSA используют дискретные логарифмы на эллиптической кривой.

Пара ключей для идентификации

В криптографии с открытым ключом каждая сторона (пользователь и удаленный хост) имеет ключ. Ключ, принадлежащий пользователю, называется **ключом пользователя**. Этот ключ используется для доступа к удаленным хостам через SSH вместо предоставления пароля. Аналогично, ключ, принадлежащий удаленному хосту, (к примеру, виртуальной машине Boot) называется **хостовым ключом**. Основное назначение этого ключа с точки зрения пользователя это проверка подлинности хоста - с его помощью вы можете быть уверены, что попадете на проверенный хост.

Использование публичного и приватного ключа для защищенного обмена данными

[пара ключей]

Оба ключа (пользовательский и хостовый) разделены на две части: **публичный ключ** и **приватный ключ**. В связи с этим их часто называют **пара ключей**. Публичная часть ключа хранит конкретное условие математической задачи, используемой выбранным алгоритмом шифрования. Приватная часть ключа хранит решение той же самой математической задачи, например, простые делители большого целого числа в алгоритме RSA. Публичная часть ключа позволяет **зашифровать** сообщение, которое сможет **расшифровать** только владелец приватного ключа.

Использование публичных и приватных ключей для цифровой подписи

[подписывание]

Приватная часть ключа помимо расшифровки сообщений также может использоваться для **подписывания** любой информации (называемой **сообщением**). Подписание (или **создание цифровой подписи**) означает генерацию набора байтов, позволяющего проверить, что конкретное сообщение было создано владельцем закрытого ключа и что оно не было изменено с тех пор. Сгенерированная подпись отправляется одновременно с сообщением, и каждый получатель, имеющий **открытый ключ отправителя**, может проверить подлинность подписи. Благодаря этим свойствам часть открытого ключа может свободно передаваться по незащищенным каналам связи. И наоборот, часть закрытого ключа должна быть доступна только ее владельцу.

Возвращаясь к конфигурации SSH, давайте взглянем на то, как обычно конфигурируются и используются пара ключей. Сгенерировать пару ключей можно командой:

Генерация пары ключей

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/myuser/.ssh/id_rsa): my_key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

```
Your identification has been saved in my_key
Your public key has been saved in my_key.pub
The key fingerprint is:
SHA256:gHexXzD2EuQFcXgsnJVTpkTKfDMSI5uFUK3J58/rjf8 my_key
The key's randomart image is:
+---[RSA 3072]----+
|      .o+=%O*oo  |
|     .  .%*%=o   |
|    . o.=oBoB.   |
|     . o+..= o   |
|        So.      |
|          .      |
|           o     |
|            oo   |
|           .+oo.E|
+----[SHA256]-----+
```

Приватный ключ сохраняется в файле под названием `my_key` в текущей директории, а публичный ключ - в файл `my_key.pub`. Эти файлы выглядят так:

Как выглядят ключи

```
# Приватный ключ
$ cat my_key
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAoi/l2u3snJTbHYgcOSSq2WHmUw9Wy10ldZgXDCgR7HahtxvWB0n0
5XurIIeUmHpfLJoinJEK4ScI7uaagtwID7QoFHsok0zlkpSMHjJLbdYAAquotjmCzBuKtG
3YMcPX0CojSCkDtFAliqUTDe+RfJ0aHVyUeqLJhCOKMEeG7UNALt0lzpUSDeVNYiBnlyd8
B+iiBWtF7MGTiMTxk8fXbrO97XF8p1PjKhCppxjocilQIwhx5HB0rJjy5vDokpvrFc6kkc
Mjff2FzLm1BbMH/WOfqNpszTRV3lK8PCAQgmWM/vBINRdY7Z6eOJVeDSvy+J+A88AjnfEq
V0MHofSoYVrals26WGasFOCboJlTgsafFWYUxOi9BGPs5Ze1kqDZmbm6l3qWGC1TD0wyWw
6Bg0XU0m5zOsmZPG6M8XbIAXWIErmxssTZHx1sUzfg5f5W7WdoWNoYqkajZY01+IB0PhCr
BxRdio4B+pR4MVV0hgto8xtWr3x9qF+wa070SodfAAAFkI5E48yOROPMAAAAB3NzaC1yc2
EAAAGBAKIv5drt7JyU2x2IHDkkqtlh5lMPVstdJXWYFwwoEex2obcb1gdJ9OV7qyCHlJh6
XyyaIpyRCuEnCO7mmoLcCA+0KBR7KJNM5ZKUjB4yS23WAAKrqLY5gswbirRt2DHD19AqI0
gpA7RQJYqlEw3vkXydGh1clHqiyYQjijBHhu1DQC7dJc6VEg3lTWIgZ5cnfAfoogVrRezB
k4jE8ZPH126zve1xfKdT4yoQqacY6HIpUCMIceRwdKyY8ubw6JKb6xXOpJHDI339hcy5tQ
WzB/1jn6jabM00Vd5SvDwgEIJljP7wSDUXWO2enjiVXg0r8vifgPPAI53xKldDB6H0qGFa
2pbNulhmrBTgm6CZU4LGnxVmFMTovQRj7OWXtZKg2Zm5upd6lhgtUw9MMlsOgYNF1NJucz
rJmTxujPF2yAF1iBK5sbLE2R8dbFM34OX+Vu1naFjaGKpGo2WNNfiAdD4QqwcUXYqOAfqU
eDFVdIYLaPMbVq98fahfsGtO9EqHXwAAAAMBAAEAAAGAZlzastGlm7HrlXj5bytwVWEPsG
6m9hVk9hI2wapsnZTGPj5oWBNaaJgkCpTnuVDKzui9XZnBhxdO8RFEhcD/qYGoJj0Q/97x
qhDtWoWdy8XcHdNf2Rr1LYNYiMYnREl55V0jBYE1YFGRUC8dlpcUeNTizZNH9xrVGvwfVJ
dgVlEyqiFTok29pl2J+JvBJcp64rb1w3vQFzyZtCGw4venRaxV/A27ghRU9JCtstPqqVrf
xCypTWeYi/LAo/d6okWbGadc1y/RJZRob437+x2HuRg+bMStrOdBU3ry4OopHoHSF6UWHa
/u9uhwbDASvXN0DgnfQe83GRTunzKDyK3zGUyRh6DpDP/btS1Y899CTE2QXIgOnSk6Ah/R
zdY9Tw8yH02SvK4Ha4E+J1VUi3dWEt1izjv+qLbAgWJqn/i585vglp5NWE0BUGg56b6dme
MbwJJ7uG1VRWNBwbzirVHmeElNo6KxvUvUYok/wNxCfKRIkpKrF+Z1cwSK8lGnVF5JAAAA
wQCRweD9plo9rpk5vKggIHwGAsFVvgUBlfybJcCrza7fdGjw2fUnpdDDGN2nltLy3D+hn/
```

```
jFeVgXBaa0TNzUSuLyKXflzuoHHimjoBGBSXCh/TLbsbebdXWr3LEorErqeBDj5NUiWdyx
TEc0qXoL3k766ZtL67mYSM45kG76FKJxWpJ3JSB9NHG4FclQBGmZ1QphMEB4Nq1RiF5yH6
ZqoguKWxmSfZ2LfLTTyR9k4TLWrIEzfMRkMCHF3fIRrTsygbkAAADBANSkkcgeiL9uBdyP
C5W9zgLDPj4wCetdqlyP3Qoi1YvKBKwBNiW60UDQ1hda75roKmIr9rfFGZVYeYk0W/8JbS
QXG9t8vtyoWghRciIYokqQ5LJpfO6fn7NOJe2t44pcQ1COux4lIlPukfMU762cXVEio8qv
zAzlC09zbXtpxyQ9W/KT4PwI0o4AlYYANiSlPRjbXm8OtvBlD9Gp46Qp4Uzx1TBnWXEpKw
E1dgbmAYuJ8NIiK7cF4TYlhRCFhJagnQAAAMEAw0Gpx6b3df776ZEDlL+9wOtiEpgNJzzI
V/O9IzDXUIeJuduLnyPocuiPWm5bxUQD/m9ZHmZjmCPkRDszgTCc1wj32Yj2/ZqUidUkOd
IlYreuzNFcfpsDj8tN3LMYcA5c+LrjYt/+pyu5lj5URKo/nMKy/2WZQW3jqZzuSt+YKTRC
dxY2eKr+4J7p3UAJ6nU4YbTqfEvk6xZktrwfmJghW7Xsmjy2XTRU3UyzBM/K9REiuekry9
+jWazSoOMcZbErAAAAFXZhbmlhLXBvb2hAaTEwNzU5Nzk5OQECAwQF
-----END OPENSSH PRIVATE KEY-----

# Публичный ключ
$ cat my_key.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCiL+Xa7eyclNsdiBw5JKrZYeZTD1bLXSV1mBcMKBHsdqG3G9YHSfTle6
sgh5SYel8smiKckQrhJwju5pqC3AgPtCgUeyiTTOWSlIweMktt1gACq6i2OYLMG4q0bdgxw9fQKiNIKQO0UCWK
pRMN75F8nRodXJR6osmEI4owR4btQ0Au3SXOlRIN5U1iIGeXJ3wH6KIFa0XswZOIxPGTx9dus73tcXynU+MqEK
mnGOhyKVAjCHHkcHSsmPLm8OiSm+sVzqSRwyN9/YXMubUFswf9Y5+o2mzNNFXeUrw8IBCCZYz+8Eg1F1jtnp44
lV4NK/L4n4DzwCOd8SpXQweh9KhhWtqWzbpYZqwU4JugmVOCxp8VZhTE6L0EY+zll7WSoNmZubqXepYYLVMPTD
JbDoGDRdTSbnM6yZk8bozxdsgBdYgSubGyxNkfHWxTN+Dl/lbtZ2hY2hiqRqNljTX4gHQ+EKsHFF2KjgH6lHgx
VXSGC2jzG1avfH2oX7BrTvRKh18= my_key
```

Ключи хоста имеют тот же формат, что и ключи пользователя. Хотя вы можете сгенерировать эти ключи вручную и указать их в конфигурации SSH-сервера, в большинстве случаев они генерируются автоматически во время первого запуска SSH-сервера и обычно находятся в файлах `/etc/ssh/ssh_host_*`. Чтобы получить доступ к хосту по SSH, вам необходимо скопировать открытые ключи каждого разрешенного пользователя в список разрешенных ключей на этом хосте. Точное расположение текстовых файлов конфигурации со списком разрешенных пользовательских ключей можно указать в настройках SSH-сервера. Обычно при работе с виртуальными машинами список разрешенных публичных ключей новой виртуальной машины указывается перед ее созданием. Имея этот список, соответствующая облачная платформа автоматически копирует все разрешенные ключи в правильный файл конфигурации внутри запускаемой виртуальной машины.

Boot работает аналогично другим облачным платформам и позволяет предоставить список разрешенных публичных ключей для каждой виртуальной машины по-отдельности. Разница состоит в том, что вам необходимо указать один и тот же авторизованный ключ в двух местах:

1. **В конфигурацию jump host.** Это необходимо для того, чтобы позволить владельцу ключа пройти через jump host и соединиться с виртуальной машиной Boot. Список разрешенных ключей для jump host указывается в файле `values.yaml` при установке Boot с помощью Helm.

   Передача списка разрешенных ключей в файле values.yaml

   ```
   boot:
   ```

```
  jumphost:
    authorizedKeys:
      - ssh-rsa AAAAB3NzaC...rTvRKh18= my_key # See public key contents above
```

Подробности описаны в разделе jump host.

2. **В настройках каждой виртуальной машины.** Это позволяет получить доступ на конкретную виртуальную машину. При создании виртуальных машин с помощью kubectl и YAML соответствующие настройки выглядят так:

   Добавление ключей в YAML описание виртуальной машины

```
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: boot
spec:
  os: ubuntu
  authorizedKeys: # Список разрешенных публичных ключей для данной виртуальной машины
    - ssh-rsa AAAAB3NzaC...rTvRKh18= my_key # Смотри пример публичного ключа выше
    - ssh-rsa AAAAB3N... another-key # Еще один публичный ключ
```

   Если виртуальная машина создается с помощью плагина kubectl - используйте опцию `-auth-keys`:

   Добавление ключей в виртуальную машину с помощью плагина kubectl

```
$ kubectl vm create -os ubuntu -auth-keys /path/to/authorized_keys.file -n boot
```

Сертификаты хоста и сертификаты пользователя

Хотя ключи пользователя и хоста остаются самой популярной технологией аутентификации SSH, у них есть несколько важных недостатков:

1. Необходимо копировать один и тот же ключ авторизованного пользователя на каждую виртуальную машину, на которую вам нужен доступ.
2. Необходимо явно доверять каждому ключу хоста перед доступом к этому хосту.
3. Ключи не хранят информацию об именах пользователей, которым разрешено аутентифицироваться с помощью этого ключа.
4. Ключи не имеют временных настроек, т. е. не имеют даты начала и окончания срока действия, поэтому их замена производится редко, либо не делается никогда.
5. Нет возможности тонкой настройки прав доступа, то есть ключи позволяют использовать любую функциональность SSH которая не запрещена явным образом: интерактивные сессии, переадресация портов, переадресация X11, возможность пробрасывать ssh ключ на другую машину.

Современные версии протокола SSH поддерживают еще одну технологию аутентификации под названием **сертификаты**. Сертификат это публичный ключ пользователя с дополнительными метаданными (имена пользователей, продолжительность сессии, дополнительные права и так далее), подписанный **удостоверяющим центром (CA)**.

Центр сертификации или удостоверяющий центр это сторона (отдел, организация), которой вы полностью доверяете. Это, к примеру, может быть отдел информационной безопасности вашей компании, либо сторонний удостоверяющий центр.

Сертификат подписанный удостоверяющим центром

[подписанный сертификат]

Работа с SSH сертификатами начинается с создания стандартной пары SSH ключей состоящих из публичного и приватного ключей. Затем публичный ключ посылается в центр сертификации который добавляет к нему цифровую подпись и необходимые метаданные. Полученный файл называется **SSH сертификатом**.

Центру сертификации должны доверять и хост и пользователь

[шифрование с помощью сертификатов]

Основная идея сертификатов заключается в том, что вместо копирования авторизованных ключей каждого пользователя мы доверяем всем сертификатам, подписанным центром сертификации, которому мы доверяем. Например, мы доверяем всем сертификатам SSH, созданным нашим отделом информационной безопасности. Единственное, что нам нужно предоставить в этом случае, это **сертификат центра сертификации**. Как и любой другой сертификат, он содержит открытый ключ и метаданные: имя центра сертификации, информацию о сроке действия и так далее. Когда мы копируем сертификат в список доверенных сертификатов хоста, этот хост затем может прочитать открытый ключ из сертификата CA и использовать его для проверки цифровой подписи на каждом сертификате, созданном этим центром сертификации. Если цифровая подпись на **сертификате пользователя** действительна, SSH-соединение пользователя с хостом разрешается.

Как и в случае с сертификатом пользователя, каждый хост может иметь **сертификат хоста**, и пользователи могут проверить, что этот сертификат был выдан доверенным центром сертификации. С точки зрения шифрования соединения сертификаты работают точно так же, как авторизованные ключи. Чтобы зашифровать и отправить некоторые данные пользователю, хост во время SSH-соединения получает сертификат от пользователя, извлекает из этого сертификата открытый ключ и использует его для шифрования. Только пользователь, имеющий закрытый ключ, соответствующий открытому ключу сертификата, может расшифровать данные.

1. В ниже мы создаем новые ключи центра сертификации. На практике ваш отдел информационной безопасности либо ответственная сторонняя организация уже имеет эти ключи. Мы демонстрируем как создавать ключи, чтобы показать полный процесс от начала до конца.
2. Мы используем алгоритм шифрования `ed25519` так как в настоящее время он считается самым надежным.

Давайте взглянем на стандартный пример использования сертификатов. Мы предполагаем что пара ключей под названием `my_key` и `my_key.pub` уже сгенерирована как описано выше. Начнем с создания ключей центра сертификации:

Создание центра сертификации

```
$ ssh-keygen -t ed25519 -f test_ca -C 'test-ca'
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in test_ca
Your public key has been saved in test_ca.pub
The key fingerprint is:
SHA256:uGmsLCSBD3Zwu05jIDMPKu+xMap+eBQTk5Tqf0enhFg test-ca
The key's randomart image is:
+--[ED25519 256]--+
|  ..o            |
| . *             |
|. + +            |
|O+ = E .         |
|=Oo * o S        |
|+.+B o = .       |
|.+@ . B o        |
| oo@ + o         |
|=o=.+ .          |
+----[SHA256]-----+
```

Как и при создании обычной пары SSH ключей, мы получим два файла:

Файлы центра сертификации

```
# Приватный ключ, держите в секрете!
$ cat test_ca
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACDdXZr6c+pc0+IZskr+Hvgm8vU5HqSPQWfokHqaRLz1PwAAAJD4P2KI+D9i
iAAAAAtzc2gtZWQyNTUxOQAAACDdXZr6c+pc0+IZskr+Hvgm8vU5HqSPQWfokHqaRLz1Pw
AAAEAEVZJTuqanwlJTfVWJsX63gLv5W3ZoDXxQXLg+HHZnTd1dmvpz6lzT4hmySv4e+Cby
9TkepI9BZ+iQeppEvPU/AAAAB3Rlc3QtY2EBAgMEBQY=
-----END OPENSSH PRIVATE KEY-----

# Публичный ключ, будет использован в настройках Boot.
$ cat test_ca.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIN1dmvpz6lzT4hmySv4e+Cby9TkepI9BZ+iQeppEvPU/ test-
ca
```

Давайте теперь подпишем существующий публичный ключ пользователя с помощью центра сертификации и создадим сертификат пользователя:

Создание сертификата пользователя

```
$ ssh-keygen -s test_ca -I 'my-test-cert' -z '0001' -n jump,root my_key.pub
Signed user key my_key-cert.pub: id "my-test-cert" serial 1 for jump,root valid
forever
```

В этом примере мы использовали приватный ключ центра сертификации `test_ca` для подписывания публичного ключа пользователя `my_key.pub`, после чего сертификат получил идентификатор `my-test-cert` (обычно используемый для входа на сервер), серийный номер `0001` и имена пользователей, авторизованных для входа на сервер - `jump` и `root`. Пользователь с именем `jump` будет авторизоваться на jump host, а пользователь с именем `root` будет авторизоваться на виртуальной машине. Итоговый сертификат сохраняется в файл `my_key-cert.pub` и выглядит так:

Обычный сертификат пользователя

```
$ cat ~/my_key-cert.pub
ssh-rsa-cert-v01@openssh.com
AAAAHHNzaC1yc2EtY2VydC12MDFAb3BlbnNzaC5jb20AAAAgUmU82rtbl76TteeXo8oVJHhMR6m6//MG/KzG5O
/3WGYAAAADAQABAAABgQCiL+Xa7eyclNsdiBw5JKrZYeZTD1bLXSV1mBcMKBHsdqG3G9YHSfTle6sgh5SYel8s
miKckQrhJwju5pqC3AgPtCgUeyiTTOWSlIweMktt1gACq6i2OYLMG4q0bdgxw9fQKiNIKQO0UCWKpRMN75F8nR
odXJR6osmEI4owR4btQ0Au3SXOlRIN5U1iIGeXJ3wH6KIFa0XswZOIxPGTx9dus73tcXynU+MqEKmnGOhyKVAj
CHHkcHSsmPLm8OiSm+sVzqSRwyN9/YXMubUFswf9Y5+o2mzNNFXeUrw8IBCCZYz+8Eg1F1jtnp44lV4NK/L4n4
DzwCOd8SpXQweh9KhhWtqWzbpYZqwU4JugmVOCxp8VZhTE6L0EY+zll7WSoNmZubqXepYYLVMPTDJbDoGDRdTS
bnM6yZk8bozxdsgBdYgSubGyxNkfHWxTN+Dl/lbtZ2hY2hiqRqNljTX4gHQ+EKsHFF2KjgH6lHgxVXSGC2jzG1
avfH2oX7BrTvRKh18AAAAAAAAAAQAAAAEAAAAMbXktdGVzdC1jZXJ0AAAAEAAAAARqdW1wAAAABHJvb3QAAAAA
AAAAAP/////////AAAAAAAAAIIAAAAVcGVybWl0LVgxMS1mb3J3YXJkaW5nAAAAAAAAABdwZXJtaXQtYWdlbn
QtZm9yd2FyZGluZwAAAAAAAAAWcGVybWl0LXBvcnQtZm9yd2FyZGluZwAAAAAAAAAKcGVybWl0LXB0eQAAAAAA
AAAOcGVybWl0LXVzZXItcmMAAAAAAAAAAAAAAAADMAAAALc3NoLWVkMjU1MTkAAAAg3V2a+nPqXNPiGbJK/h74Jv
L1OR6kj0Fn6JB6mkS89T8AAABTAAAAC3NzaC1lZDI1NTE5AAAAQDoV3uwn0SBWrZcHtnzNuVVYPXe3orfxb3gp
M6jhzszPaZ96G5Pu48lyQPN9tDiZE1PJJ7e/ovw07LQy4wqkSAo= some-comment-here
```

Теперь, перед началом работы с Boot нужно добавить публичный ключ центра сертификации в два места:

1. **В конфигурацию jump host.** Это необходимо для того, чтобы позволить владельцу SSH сертификата пройти авторизацию в jump host. Список доверенных сертификатов пользователя предоставляется в файле `values.yaml` при установке Boot с помощью Helm:
2. Добавление центров сертификации в конфигурацию jump host в values.yaml

```
boot:
  jumphost:
    trustedUserCAKeys:
      - ssh-ed25519 AAAAC3NzaC1....eppEvPU/ test-ca # Сюда вставляем публичную часть
сертификата центра сертификации
```

+ Детально это описано в разделе конфигурация jump host.

1. **В каждую виртуальную машину.** Это позволяет получить доступ к конкретной виртуальной машине. При создании виртуальной машины с помощью kubectl и YAML нужное описание виртуальной машины выглядят так:

   Добавление ключей в YAML описание виртуальной машины

   ```
   apiVersion: boot.aerokube.com/v1
   kind: VirtualMachine
   metadata:
     name: my-vm
     namespace: boot
   spec:
     os: ubuntu
     trustedUserCAKeys: # Список разрешенных центров сертификации для доступа на
   виртуальную машину
       - ssh-ed25519 AAAAC3NzaC1....eppEvPU/ test-ca # Сюда вставляем публичную часть
   сертификата центра сертификации
       - ssh-ed25519 AAAAC3.... another-ca # Другой центр сертификации
   ```

   Если виртуальная машина создается с помощью плагина для kubectl - используйте опцию `-user-ca-keys`:

   Добавление центров сертификации в виртуальную машину с помощью плагина kubectl plugin

   ```
   $ kubectl vm create -os ubuntu -user-ca-keys /path/to/truster_user_ca.file -n boot
   ```

3.2.3. Доступ к виртуальной машине

Теперь, когда у нас есть сконфигурированный jump host и виртуальная машина под названием `my-vm` - давайте зайдем на нее.

Обычная виртуальная машина

```
$ kubectl get vm -n boot
NAME    FQDN                                IP           OS             CPU    MEM
STATUS    UPTIME
my-vm   my-vm.vm.boot.svc.cluster.local   172.17.0.4   ubuntu:22.04   250m   1Gi
Running   10s
```

SSH команда для доступа через jump host с нужными аргументами выглядит так:

Доступ к виртуальной машине через jump host

```
$ ssh -i /path/to/my_key -J jump@boot.example.com:2222 root@my-
vm.vm.boot.svc.cluster.local
```

Аргумент `-i` используется для использования приватного ключа из файла, аргумент `-J` - для использования jump host, строка `jump@boot.example.com:2222` означает что соединение

проходит через jump host `boot.example.com` на порту `2222` используя имя пользователя `jump` (это настройки Boot по умолчанию). Наконец, мы указываем хост назначения `root@my-vm.vm.boot.svc.cluster.local`, то есть соединяемся к хосту внутри Kubernetes с доменным именем `my-vm.vm.boot.svc.cluster.local` используя имя пользователя `root`. Эта команда будет работать как для разрешенных ключей, так и для сертификатов. В случае сертификатов SSH клиент автоматически переберет все сертификаты соответствующие приватному ключу. Конечно, вы можете указать и конкретный сертификат командой:

Предоставление файла сертификата пользователя

```
$ ssh -o CertificateFile=/path/to/my_key-cert.pub -i /path/to/my_key -J
jump@boot.example.com:2222 root@my-vm.vm.boot.svc.cluster.local
```

Команды выглядят на первый взгляд сложными, не так ли? Но хорошая новость заключается в том что параметры этих команд, за исключением названий виртуальных машин, меняются редко и соответственно эти настройки можно сохранить в конфигурационном файле SSH клиента (обычно в директории `~/.ssh/config`):

Добавление настроек в конфигурацию SSH клиента

```
$ cat ~/.ssh/config
Host bootjump
    IdentityFile /path/to/my_key # Расположение приватного ключа
    HostName boot.example.com # Имя хоста jump host
    Port 2222 # Сетевой порт jump host
Host *.vm.boot.svc.cluster.local
    IdentityFile /path/to/my_key # Расположение приватного ключа
    ProxyJump jump@bootjump # Всегда использовать jump host, описанный выше
```

После этого изменения подключение к виртуальной машине выглядит проще:

Упрощенный доступ к виртуальной машине

```
$ ssh root@my-vm.vm.boot.svc.cluster.local
```

Однако, можно сделать команду еще короче. Доменное имя виртуальной машины - `my-vm.vm.boot.svc.cluster.local`. `boot` - это имя неймспейса, а `svc.cluster.local` - стандартный DNS суффикс в Kubernetes. Kubernetes может автоматически добавлять эту часть доменного имени так что можно добавить в настройки SSH клиента следующую информацию:

Поддержка короткого доменного имени

```
$ cat ~/.ssh/config
Host *.vm
    IdentityFile /path/to/my_key # Расположение приватного ключа
    ProxyJump jump@bootjump
```

После этого команда выглядит так:

Еще более упрощенная команда

```
$ ssh root@my-vm.vm
```

3.2.4. Добавление хоста Boot в список доверенных хостов

Как упоминалось ранее, сертификат могут иметь как пользователь, так и хост. Сертификаты пользователя используются SSH сервером для аутентификации. Сертификаты хоста дают гарантию пользователям в том, что соединение было установлено именно с тем хостом с каким планировалось. По умолчанию при первом соединении с виртуальной машиной вы видите следующее сообщение:

Первое соединение с виртуальной машиной

```
$ ssh root@my-vm.vm
The authenticity of host '[boot.example.com]:2222 ([XXX.XXX.XXX.XXX]:2222)' can't be
established.
ED25519 key fingerprint is SHA256:F/qgcVr/35EPaGtxX4Y1YHR5H8oD28sUKs0IRFiRWfI.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[boot.aerokube.local]:2222' (ED25519) to the list of known
hosts.
The authenticity of host 'my-vm.vm (<no hostip for proxy command>)' can't be
established.
ED25519 key fingerprint is SHA256:mGSVISH1obExpV3YceqKuzVrIzbnv7vwiUjc9CccS1k.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'my-vm.vm' (ED25519) to the list of known hosts.
my-vm:~#
```

Чтобы убрать это предупреждение, необходимо добавить виртуальную машину Boot в список доверенных хостов. При установке Boot автоматически создает центр сертификации, который используется для генерации хостовых сертификатов для всех создаваемых виртуальных машин. Пара ключей для этого сертификационного центра хранится в секрете Kubernetes `<boot-namespace>-vm-ssh-host-ca-keys`, например, если вы запускаете Boot в дефолтном неймспейсе `boot` - его имя будет `boot-vm-ssh-host-ca-keys`.

1. Извлеките открытый ключ центра сертификации из секрета:

   ```
   $ kubectl get secret boot-vm-ssh-host-ca-keys -o jsonpath='{.data.ca_key\.pub}' -n
   boot | base64 -D
   ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIFHZc6j06RCf+8/J1IuS8SY1/hasLHlik6XRnO87+Zhu
   user@boot-generate-ssh-keys-4j2dw
   ```

2. Добавьте этот ключ в SSH known hosts:

   ```
   $ cat ~/.ssh/known_hosts
   ```

```
# Для полного доменного имени
@cert-authority *.vm.boot.svc.cluster.local ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIFHZc6j06RCf+8/J1IuS8SY1/hasLHlik6XRnO87+Zhu user@boot-
generate-ssh-keys-4j2dw

# Для сокращенного доменного имени
@cert-authority *.vm ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIFHZc6j06RCf+8/J1IuS8SY1/hasLHlik6XRnO87+Zhu user@boot-
generate-ssh-keys-4j2dw
```

3. После выполнения этих настроек SSH больше не будет просить вас подтвердить аутентификацию при логине на виртуальные машины. Однако, при первом использовании вам будет необходимо подтвердить вход на jump host (`boot.example.com`).

4. Настройка

4.1. Операционные системы

1. Виртуальные машинны Boot всегда запускаются в привилегированном режиме, поэтому мы намеренно не добавляли уже существующие функциональности Kubernetes, например security context.
2. По умолчанию, подключение к виртуальным машинам делается по SSH. Интерактивная SSH сессия никогда не использует серверные переменные окружения, поэтому мы не добавляли реализацию выставления переменных окружения environment variables поду.

Операционная система - основной объект в Boot. Операционная система используются при запуске виртуальной машины. Из нее берутся: образ контейнера, доступные вычислительные ресурсы по-умолчанию, конфигурация DNS и так далее. Для вывода списка операционных систем выполните команду:

Вывод списка операционных систем

```
$ kubectl get os -n boot
NAME     REPOSITORY                          VERSION(S)   CPU    MEM   AGE
alpine   registry.aerokube.ru/boot/alpine    3.18         500m   1Gi   50m
ubuntu   registry.aerokube.ru/boot/ubuntu    22.04        500m   1Gi   50m
```

Информацию о конфигурации операционной системы можно получить командой:

Получение описания операционной системы в формате YAML

```
$ kubectl get os ubuntu -n boot -o yaml
apiVersion: boot.aerokube.com/v1
kind: OperatingSystem
metadata:
  name: ubuntu
```

```
  namespace: boot
  # Другие метаданные
spec:
  repository: registry.aerokube.ru/boot/ubuntu  ①
  resources:                                    ②
    limits:
      cpu: "0.5"
      memory: 1Gi
    requests:
      cpu: "0.5"
      memory: 1Gi
  versions:                                     ③
  - "22.04"
status:
  # Поля, используемые для отображения списка
```

① Репозиторий с образами операционной системы

② Доступные вычислительные ресурсы

③ Доступные версии операционной системы

Этой простой конфигурации хватит для успешной работы, но существуют и более продвинутые настройки:

Другие поля в описании операционной системы

```
$ kubectl get os ubuntu -n boot -o yaml
apiVersion: boot.aerokube.com/v1
kind: OperatingSystem
metadata:
  name: ubuntu
  namespace: boot
  # Другие метаданные
spec:
  affinity:                                 ①
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: node-label-1
            operator: In
            values:
            - value-1
            - value-2
  dnsPolicy: None                           ②
  dnsConfig:                                ③
    nameservers:
      - 192.0.2.1
    searches:
      - ns1.svc.cluster-domain.example
      - my.dns.search.suffix
```

```
    options:
      # Смотри документацию Kubernetes
  nodeSelector:                          ④
      node-label-1: "label1-value"
      node-label-2: "label2-value"
  repository: registry.aerokube.ru/boot/ubuntu      ⑤
  resources:                             ⑥
    limits:
      cpu: "0.5"
      memory: 1Gi
    requests:
      cpu: "0.5"
      memory: 1Gi
  tolerations:                           ⑦
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
  versions:                              ⑧
  - "22.04"
status:
  # Fields used for listing
```

① Настройки affinity

② Настройки политики DNS

③ Настройки конфигурации DNS

④ Настройки Kubernetes node selector

⑤ Репозиторий с образами операционной системы - единственное поле обязательное для заполнения

⑥ Настройки вычислительных ресурсов

⑦ Настройки Kubernetes tolerations

⑧ Доступные версии операционной системы - ограничивает теги образов, которые можно использовать для запуска виртуальных машин

Далее эти настройки описаны более подробно.

Изменить настройки операционной системы:

Редактирование операционной системы

```
$ kubectl edit os ubuntu -n boot # Внесите изменения в текстовом редакторе, сохраните
и выйдите из текстового редактора
```

Другой, рекомендуемый нами, способ переопределить значения операционной системы, который будет работать в большинстве случаев - использование Helm. Переопределите значения в файле `values.yaml`:

Редактирование операционной системы с помощью Helm

```
boot:
  # Тут находятся другие поля
  os:
    ubuntu: # Поля в Helm чарте ровно такие же как описаны выше
      repository: registry.aerokube.ru/boot/ubuntu
      versions:
      - "22.04"
    nodeSelector: # Например, можно добавить node selector
      kubernetes.io/arch: "amd64"
  # Другие поля
```

Если применить Helm чарт заново с новыми значениями из `values.yaml`, настройки операционной системы поменяются.

4.1.1. Версии и репозитории

Самые важные настройки в описании операционной системы это имя репозитория и доступные версии образа. К примеру, ваша операционная система имеет следующее описание:

Репозиторий и версии в описании операционной системы

```
$ kubectl get os ubuntu -n boot -o yaml
apiVersion: boot.aerokube.com/v1
kind: OperatingSystem
metadata:
  name: ubuntu
  namespace: boot
  # Другие поля метаданных
spec:
  repository: registry.aerokube.ru/boot/ubuntu
  versions:
  - "20.04"
  - "22.04"
status:
  # Поля используемые для вывода списка
```

Например, мы создаем виртуальную машину с таким определением:

Запрос на создание виртуальной машины с Ubuntu 20.04

```
apiVersion: boot.aerokube.com/v1
kind: VirtualMachine
metadata:
  name: my-ubuntu-vm
  namespace: boot
spec:
  os: ubuntu
```

```
  version: "20.04"
```

Boot создаст виртуальную машину, используя образ контейнера `registry.aerokube.ru/boot/ubuntu:20.04`. Если поменять версию на `22.04` соответствующий образ контейнера будет `registry.aerokube.ru/boot/ubuntu:22.04`. Список версий нужен в основном для того, чтобы показывать пользователю рекомендуемые версии операционных систем. Ничто не мешает пользователю создать виртуальную машину с версией `18.04`, конечно, если образ `registry.aerokube.ru/boot/ubuntu:18.04` существует.

4.1.2. Вычислительные ресурсы

1. Boot использует такое же описание выделяемых вычислительных ресурсов в формате YAML, как и Kubernetes.

Для того чтобы задать лимиты вычислительных ресурсов для каждой виртуальной машины, вам нужно добавить несколько строк в файл описания операционной системы.

Настройка вычислительных ресурсов в описании операционной системы

```
boot:
  # Другие поля
  os:
    ubuntu:
      repository: registry.aerokube.ru/boot/ubuntu
      resources: # Настройка вычислительных ресурсов операционной системы
        limits:
          cpu: "1.0"
          memory: 1Gi
        requests:
          cpu: "0.5"
          memory: 1Gi
  # Другие поля
```

4.1.3. Node Selector

Boot использует тот же формат описания node selector в формате YAML, что и Kubernetes.

Иногда возникает необходимость запускать виртуальные машины только на определенных нодах Kubernetes. Kubernetes позволяет это сделать благодаря механизму node selector. Если вам необходимо определить node selector в операционной системе - добавьте строчку в `values.yaml`:

Добавление node selector

```
boot:
  # Другие поля
  os:
```

```
    ubuntu:
      repository: registry.aerokube.ru/boot/ubuntu
      nodeSelector: # Определение node selector
        node-label-1: "label1-value"
  # Другие поля
```

4.1.4. Affinity

Boot использует то же описание affinity в формате YAML, что и Kubernetes.

В дополнение к управлению node selector, вы можете использовать возможность настройки affinity доступные в Kubernetes. Это позволяет вам иметь еще более продвинутые настройки планировщика, такие как сопоставление нод Kubernetes со сложными логическими выражениями, предотвращение запуска некоторых помеченных виртуальных машин на одном ноде с другими помеченными виртуальными машинами и так далее.

Настройка affinity в values.yaml

```
boot:
  # Другие поля
  os:
    ubuntu:
      repository: registry.aerokube.ru/boot/ubuntu
      affinity: # Настройки affinity имеют ровно такой же синтаксис YAML, как и
Kubernetes
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: node-label-1
                operator: In
                values:
                - value-1
                - value-2
  # Другие поля
```

4.1.5. Сетевая конфигурация

В некоторых случаях необходимо иметь гибкую сетевую конфигурацию. Например, вам может понадобиться переопределить DNS сервер. Это может быть сделано так:

Добавление DNS сервера в описание операционной системы

```
boot:
  # Другие поля
  os:
    ubuntu:
      repository: registry.aerokube.ru/boot/ubuntu
      dnsPolicy: None # Настройки DNS имеют ровно такой же синтаксис YAML, как и
```

```
Kubernetes
      dnsConfig:
        nameservers:
          - 192.0.2.1
        searches:
          - ns1.svc.cluster-domain.example
          - my.dns.search.suffix
        options:
          # Смотри документацию Kubernetes
  # Другие поля
```

4.1.6. Tolerations

Boot использует ровно такой синтаксис описания tolerations в формате YAML, что и Kubernetes.

В дополнение к node selector и affinity в Kubernetes реализован механизм tolerations. Этот механизм позволяет не размещать на указанных нодах рабочую нагрузку (pod'ы) без соответствующих "разрешений". Если вы хотите запускать виртуальные машины на таких нодах, вам нужно добавить настройку **toleration** в описание операционной системы:

Добавление tolerations в описание операционной системы

```
boot:
  # Другие поля
  os:
    ubuntu:
      repository: registry.aerokube.ru/boot/ubuntu
      tolerations: # Настройки tolerations имеют ровно такой же синтаксис YAML, как и
Kubernetes
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoSchedule"
  # Другие поля
```

4.2. Настройки jump host

Jump host - главная точка входа на виртуальные машины Boot. Jump host настраивается путем переопределения значений в файл `values.yaml` и применения Helm чарта.

4.2.1. Пользователь

Процессы в jump host всегда запускаются от непривелигированого пользователя (по умолчанию это пользователь с именем `jump`). Это сделано из соображений безопасности. Пользователи Boot могут авторизоваться на jump host только с помощью этого пользователя по протоколу SSH. Изменить или определить имя пользователя можно так:

Обновляем настройки пользователя jump host

```
$ cat values.yaml
boot:
  jumphost:
    user:
      name: my-user # Логин пользователя
      uid: 10000000 # UID пользователя
      gid: 10000000 # GID пользователя
```

4.2.2. Сетевой порт

Изменение сетевого порта для доступа на jump host:

Обновление порта

```
$ cat values.yaml
boot:
  jumphost:
    port: 22222 # Устанавливаем собственный номер порта
```

4.2.3. Разрешенные пользовательские ключи и сертификаты

Важным шагом является настройка пользователей, у которых вообще есть доступ по SSH к кластеру Boot. Пользователи настраиваются путем добавления публичных SSH ключей и доверенных центров сертификации в конфигурацию jump host:

Добавление пользователей и сертификатов

```
$ cat values.yaml
boot:
  jumphost:
    authorizedKeys: # Разрешенные ключи пользователей
      - ssh-rsa AAAAB3N... some-key
      - ssh-rsa AAAAB3N... another-key
    trustedUserCAKeys: # Доверенные центры сертификации
      - ssh-ed25519 AAAAC3N... certification-authority-1
      - ssh-ed25519 AAAAC3N... certification-authority-2
```

4.2.4. Вычислительные ресурсы

Если по какой-либо причине вычислительных ресурсов, выделяемых по умолчанию, недостаточно:

Обновление вычислительных ресурсов jump host

```
$ cat values.yaml
boot:
  deployment:
```

```
    containers:
      jumphost:
        resources: # Настройки вычислительных ресурсов jump host
          limits:
            cpu: 2000m
            memory: 1Gi
          requests:
            cpu: 200m
            memory: 1Gi
```

4.3. Лицензионный ключ

1. Согласно лицензионному соглашению вам доступен **один процессор** бесплатно и без ограничений по времени. Если вы хотите использовать больше процессоров - требуется приобрести лицензионный ключ. Этот раздел описывает как получить и установить лицензионный ключ. Пробный ключ с ограниченным сроком действия может быть сгенерирован на сайте Boot.

Лицензионный ключ представляет собой текстовый файл с расширением **.key**, который обычно выглядит так:

```
$ cat license.key
Ulh3SnN6S3JYTVg5UmIzaHNiWUpOaE1rcGltQzJxRVZVbGdHMVliNlZDbnNjVkc5b1M1eGNEbTRZYkN6c2RaTm
taaGs0cDExQWRlOTA2YVNxK3NNV2JORHd0NkFEUEZTNk16UXVCcWhQMVovajhhdWlJZDJzdW9yVEFRTFppSnp2
NHloMkdZYXNVVlNhRk05Q2ZOUk4rd1JCNHlXRlFwRmNwbVRFWk9hdXRQWjJvVUM0TldGdXR2OUtiangrT0hkRm
JNK2xtQUhCYVArWDhlUTJoNnFzRlExdHl2Zm11QmtVWUNhRHBSTEhzVTdLQXVEZFZKWlhUSU9PRjNjUlFIODhy
YmZKTkZVWm1sNG5UZnJHM2RFRTJmYkMyakNwVndLWmJaMkgrVi9zeGRXd0dDblZMNFAyYXVyNjQ4cDhnb0xvRG
dZMGlnRmM1WXFPODVGK0U4TlRPWWpyMGtPRThnY1lRcU1JT1JWZEkwQ0ZNVkk3SkFpbHI0UzhHcHduY2Vwcks3
ZERtbnVLNmRIeGVnZHhqSGNIN1laZlR6U2prZ001S2R5Q1RCSlF0RXB2VjkvUlF5MUV3M3RIcCtTRWcvTjl3eU
F5VE4xUFl4Q0xtU2t0QjFNblZVeURZby9sWXlCYlQrSGNlSUExdktTVThDQlZFaFNRZmdRS1BmbUxFblBuSmxh
VHZmWXhnVUF3b3B3dmFwaHFmaExRNTVEM2d6RzA4ZDlsNTVGVGE3Vlo4b2Vpd2FabUFDWHZRZ3NlMTUzT29SdD
V1M1VsNGNVTmFUOGUxbWgva2JKajJ1Mjk2cysvalBBa3JVRnNVdWlNZHA1a2Zrc2hTQlhybDZyVlFJYytDcWE3
MUFBdWpyT1lPNm1JZ3BNZXAvYUI4cXhRR29uTUVzVGRrRlVKR289O2V5SnNhV05sYm5ObFpTSTZJa1JsWm1GMW
JIUWlMQ0p3Y205a2RXTjBJam9pUW05dmRDSXNJbTFoZUZObGMzTnBiMjV6SWpveGZRPT0=
```

Boot хранит лицензионные ключи (или просто лицензии) в ресурсах Kubernetes.

4.3.1. Вывод списка лицензий

В отличие от других ресурсов Boot, лицензии являются ресурсами, которые хранятся **на уровне кластера** (не на уровне неймспейса). Таким образом вам не нужно указывать имя неймспейса в команде для вывода списка лицензий (опция `-n boot` не нужна).

Вывод списка доступных лицензий:

Список лицензий

```
$ kubectl get licenses
NAME   LICENSEE   CPUS   USAGE   EXPIRES   STATUS   NAMESPACE   AGE
boot   Default    1      0%      Never     Ok       boot        2m42s
```

Такой вывод вы увидите если установлена бесплатная лицензия. Значение колонок следующее:

- **Name**. Название лицензии.
- **Licensee**. Имя владельца лицензии. Обычно соответствует названию компании, например `Acme LLC`. Для бесплатных лицензий с ограничением в 1 процессор имя будет `Default`.
- **CPUs**. Максимальное количество процессоров.
- **Expires**. Количество дней до окончания действия лицензии. Если лицензия истекла - значение будет `Already`, если лицензий без срока действия - значение будет `Never`.
- **Status**. Статус лицензии. Может быть либо: `Ok` - лицензия активна, `Expired` - лицензия закончилась, `Broken` - предоставлены неверные данные лицензионного ключа.
- **Namespace**. Имя неймспейса Kubernetes, где используется данная лицензия.

Вы можете иметь несколько ресурсов Kubernetes с названием `license`. В этом случае для вывода списка лицензий Boot используйте полное имя ресурса:

Вывод списка лицензий с использованием полного имени ресурса

```
$ kubectl get licenses.boot
NAME   LICENSEE   CPUS   USAGE   EXPIRES   STATUS   NAMESPACE   AGE
boot   Default    1      0%      Never     Ok       boot        2m42s

$ kubectl get licenses.boot.aerokube.com
NAME   LICENSEE   CPUS   USAGE   EXPIRES   STATUS   NAMESPACE   AGE
boot   Default    1      0%      Never     Ok       boot        2m42s
```

Чтобы посмотреть лицензию в формате YAML:

Вывод лицензии в формате YAML

```
$ kubectl get license boot -o yaml
apiVersion: boot.aerokube.com/v1
kind: License
metadata:
  name: boot                        ①
  # Другие поля метаданных
spec:
  data: YkV0dmNsaGxTak51Y2....      ②
status:
  # Другие ключ и значения
```

① Имя лицензии, соответствует имени неймспейса, где используется лицензия

② Содержание лицензии

4.3.2. Обновление лицензионного ключа

Для обновления лицензионного ключа достаточно отредактировать поле `data` в соответствующем объекте лицензии:

Обновление лицензии с помощью kubectl

```
$ kubectl edit license boot # Обновите поле data в текстовом редакторе, сохраните и
выйдите из редактора
```

Когда вы обновляете лицензию, все изменения применяются немедленно. После этого поды Boot поды перезапускаются, чтобы прочитать новую лицензию. Уже созданные виртуальные машины продолжают работать непрерывно.

4.3.3. Несколько лицензионных ключей

Boot позволяет использовать один лицензионный ключ сразу на несколько неймспейсов Kubernetes и в большинстве случаев одной лицензии достаточно. Однако, в некоторых случаях вам может понадобиться использовать независимые копии Boot для каждой команды и отдельный лицензионный ключ для из них:

1. Установите два независимых кластера Boot в неймспейсы `ns1` и `ns2`
2. Создайте два объекта лицензии с полем `name` со значениями `ns1` и `ns2` и сохраните файл (например `license-keys.yaml`):

   Создание лицензий

   ```
   $ cat license-keys.yaml
   apiVersion: boot.aerokube.com/v1
   kind: License
   metadata:
     name: ns1
   spec:
     data: <license-key-1>
   ---
   apiVersion: boot.aerokube.com/v1
   kind: License
   metadata:
     name: ns2
   spec:
     data: <license-key-2>
   ```

3. Примените полученный файл:

```
$ kubectl apply -f license-keys.yaml
```

- При попытке создать две лицензии с одинаковым значением в поле `data`, одна из них будет считаться дубликатом и будет автоматически удалена.
- Если у вас две лицензии с одинаковым значением в поле `namespace` - Boot всегда будет выбирать последнюю созданную лицензию.

4. Лицензионный ключ будет установлен и вы увидите следующее в списке лицензий:

Установлено два лицензионных ключа

```
$ kubectl get licenses
NAME  LICENSEE   CPUS   USAGE   EXPIRES   STATUS   NAMESPACE   AGE
ns1   Acme Inc.  10     0%      20d       Ok       ns1         2m42s
ns2   Acme Inc.  20     0%      30d       Ok       ns2         2m42s
```

4.3.4. Удаление лицензионного ключа

Чтобы удалить установленный лицензионный ключ, просто удалите его объект:

Удаление лицензии

```
$ kubectl delete license boot
```

Если вы удаляете последний лицензионный ключ со значением в поле `name` указывающим на какой-либо неймспейс Boot, то лицензия автоматически переключится на бесплатную с ограничением в один процессор.

4.3.5. Окончание срока действия лицензии

Для вывода списка лицензий с заканчивающимся или уже истекшим сроком действия используйте `kubectl`:

Вывод лицензий с истекшим сроком действия

```
$ kubectl get licenses
NAME  LICENSEE   CPUS   USAGE   EXPIRES   STATUS   NAMESPACE   AGE
ns1   Acme Inc.  10     0%      32d       Ok       ns1         2m42s
ns2   Acme Inc.  20     0%      today     Ok       ns2         2m42s
```

Колонка **Expires** показывает количество дней до окончания срока действия каждой лицензии. Когда срок действия истек, вы увидите такой вывод:

Срок действия одной из лицензий истек

```
$ kubectl get licenses
```

```
NAME  LICENSEE   CPUS   USAGE   EXPIRES   STATUS   NAMESPACE   AGE
ns1   Acme Inc.  10     0%      32d       Ok       ns1         2m42s
ns2   Acme Inc.  20     0%      Already   Expired  ns2         2m42s
```

Статус той лицензии, срок действия которой истёк, будет показан в колонке **Expires** со значением **Already**, а в колонке **Status** будет написано **Expired**.

Для ввода списка истекающих или истекших лицензий вы также можете использовать Kubernetes API напрямую, а не `kubectl`.

4.4. Использование собственного репозитория образов

По умолчанию образы контейнеров Boot (`aerokube/boot`, `aerokube/jumphost` и так далее) загружаются с публичного репозитория образов. Необходимость настроить Boot для работы с собственным репозиторием контейнеров может возникнуть, если из соображений безопасности в вашей компании образы контейнеров могут загружаться только из собственных репозиториев (к примеру `my-registry.example.com`). Это делается так:

1. Настройка аутентификации Kubernetes для доступа в приватный реестр:

```
$ kubectl create secret docker-registry my-registry.example.com --docker-server=my
-registry.example.com --docker-username=some-user --docker-password=registry
-password --docker-email=some-user@example.com -n boot
$ kubectl patch serviceaccount boot -p '{"imagePullSecrets": [{"name": "my-
registry.example.com"}]}' -n boot # Используйте правильное имя для service account
```

2. Скопируйте образы виртуальных машин в ваш репозиторий:

```
quay.io/boot/ubuntu:22.04 => my-registry.example.com/boot/ubuntu:22.04
```

3. Обновите настройки операционной системы в Helm чарте:

Настройки операционной системы в Helm чарте

```
$ cat values.yaml
boot:
  namespaces:
  os:
    ubuntu:
      repository: my-registry.example.com/boot/ubuntu
      versions:
      - "22.04"
```

4. Скопируйте нужную версию образов Boot в ваш репозиторий:

```
aerokube/boot:1.0.0 => my-registry.example.com/aerokube/boot:1.0.0
aerokube/jumphost:1.0.0 => my-registry.example.com/aerokube/jumphost:1.0.0
aerokube/keygen:1.0.0 => my-registry.example.com/aerokube/keygen:1.0.0
aerokube/reloader:1.0.0 => my-registry.example.com/aerokube/reloader:1.0.0
```

5. Для запуска Boot используйте новые образы, созданные на предыдущем этапе:

Собственный репозиторий для образов Boot в Helm чарте

```
$ cat values.yaml
boot:
  deployment:
    containers:
      boot:
        image:
          repository: my-registry.example.com/aerokube/boot
      jumphost:
        image:
          repository: my-registry.example.com/aerokube/jumphost
      keygen:
        image:
          repository: my-registry.example.com/aerokube/keygen
      reloader:
        image:
          repository: my-registry.example.com/aerokube/reloader
```

4.5. Использование нескольких неймспейсов

Этот раздел показывает как можно сконфигурировать Boot для использования виртуальных машин в нескольких неймспейсах.

1. Создайте один или несколько неймспейсов, где будут запускаться виртуальные машины.

Создание неймспейсов для каждой команды

```
$ kubectl create namespace team-alpha
$ kubectl create namespace team-beta
```

При необходимости вы можете сконфигурировать использование вычислительных ресурсов для каждого неймспейса используя механизм ограничения ресурсов в Kubernetes.

2. Создайте роли пользователей для каждого неймспейса:

Создание ролей

```
$ cat team-alpha-roles.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: team-alpha-roles
  namespace: team-alpha # Use correct namespace name here
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - watch
  - list
  - create
  - update
  - patch
  - delete
- apiGroups:
  - boot.aerokube.com
  resources:
  - operatingsystems
  - operatingsystems/status
  verbs:
  - get
  - watch
  - list
- apiGroups:
  - boot.aerokube.com
  resources:
  - virtualmachines
  - virtualmachines/status
  verbs:
  - get
  - watch
  - list
  - create
  - update
  - patch
  - delete

$ kubectl apply -f team-alpha-roles.yaml # Create role
```

3. Назначьте роли, созданные на предыдущем этапе, каждому пользователю, чтобы дать возможность создавать виртуальные машины:

   Назначение роли

```
$ kubectl create rolebinding my-user-roles --role=team-alpha-roles --user=my-user
-n team-alpha # Добавить роль пользователю my-user (используйте нужный логин)
```

4. Добавьте полученный неймспейс в файл values.yaml:

   Добавление неймспейсов в values.yaml

   ```
   $ cat values.yaml
   boot:
     namespaces:
       - team-alpha
       - team-beta
   ```

5. Примените Helm чарт:

   Применение настроек неймспейсов

   ```
   $ helm upgrade --install -f values.yaml -n boot boot aerokube/boot
   ```

6. Теперь пользователи смогут создавать виртуальные машины в своих неймспейсах:

   Создание виртуальной машины с помощью описания YAML:

   ```
   $ cat vm.yaml
   apiVersion: boot.aerokube.com/v1
   kind: VirtualMachine
   metadata:
     name: my-vm
     namespace: team-alpha # Note namespace name here
   spec:
     os: ubuntu
     authorizedKeys:
       - ssh-rsa AAAAB3N... some-key

   $ kubectl create -f ~/vm.yaml
   virtualmachine.boot.aerokube.com/my-vm created
   ```

   Создание неймспейса с помощью плагина для kubectl:

   ```
   $ kubectl vm create my-vm -os ubuntu -n team-beta # Как и в любой другой команде
   kubectl, флаг -n определяет название неймспейса
   ```

4.6. Лог файлы

Обычно Boot просто работает из коробки, но иногда возникает необходимость посмотреть лог файлы. Каждый компонент Boot пишет логи в стандартный поток вывода (`stdout`), так что вы можете использовать известные команды `kubectl`, чтобы просматривать логи. Все, что связано с запуском виртуальных машин выводится в контейнер `boot`:

```
$ kubectl logs -lapp=boot -c boot -n boot
```

Следить за логом во время запуска тестов можно с помощью опции `-f`:

```
$ kubectl logs -f -lapp=boot -c boot -n boot
```

Лог файлы контейнеров `reloader` и `jumphost` можно посмотреть таким образом:

```
$ kubectl logs -f -lapp=boot -c reloader -n boot
$ kubectl logs -f -lapp=boot -c jumphost -n boot
```

Если возникают какие-либо проблемы с виртуальными машинами - проверьте логи соответствующего пода:

```
$ kubectl logs virtual-machine-01 -n boot
```

Здесь `virtual-machine-01` это название виртуальной машины.