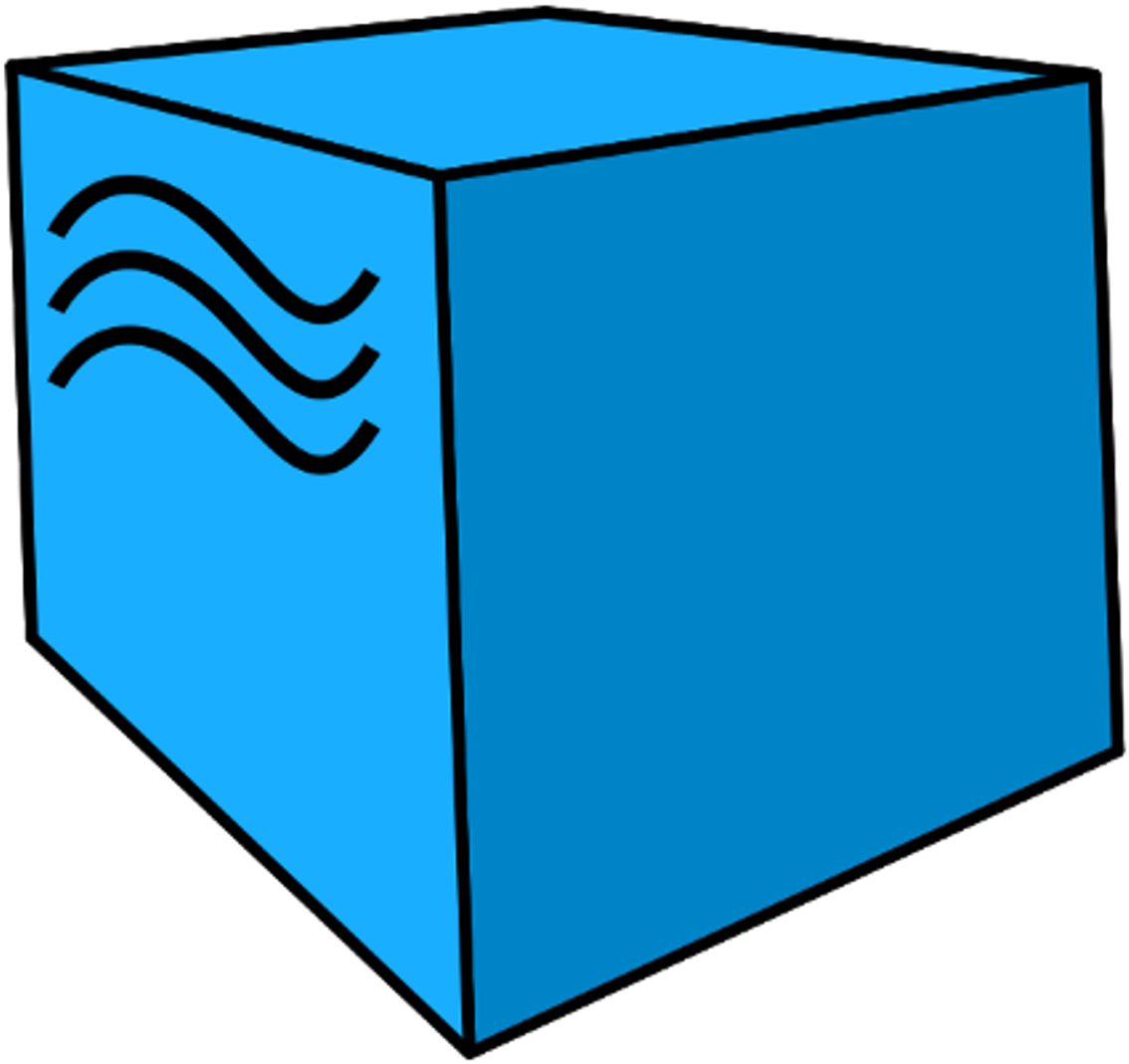
Aerokube Moon



https://aerokube.ru/

Общее Руководство по ПО "Moon"

Содержание

1. О документе . . . 3
2. Начало работы . . . 3
   2.1. Начало работы . . . 3
      2.1.1. Установка в Kubernetes . . . 3
      2.1.2. Установка в Openshift . . . 8
      2.1.3. Рекомендации по Работе с Helm . . . 9
   2.2. Архитектура . . . 11
      2.2.1. Компоненты Moon . . . 11
      2.2.2. Режимы работы Moon . . . 12
      2.2.3. Содержимое браузерных подов . . . 12
   2.3. Рекомендуемые настройки для кластера . . . 13
   2.4. Необходимые права доступа . . . 13
      2.4.1. Режим одного неймспейса . . . 13
      2.4.2. Режим множества неймспейсов . . . 14
   2.5. Различие версий Moon 2.x и Moon 1.x . . . 14
   2.6. Moon и другие решения . . . 17
3. Основная функциональность . . . 18
   3.1. Веб интерфейс . . . 18
      3.1.1. Консоль . . . 21
   3.2. Работа с Selenium . . . 24
      3.2.1. Дополнительные Selenium capability, поддерживаемые Moon . . . 25
      3.2.2. Headless режим . . . 32
      3.2.3. Запись видео . . . 32
      3.2.4. Эмуляция мобильных устройств . . . 33
      3.2.5. Доступ к буферу обмена . . . 34
      3.2.6. Загрузка файлов в браузер . . . 35
      3.2.7. Дополнительные данные для браузера . . . 36
      3.2.8. Доступ к загруженным из браузера файлам . . . 40
      3.2.9. Доступ к инструментам разработчика . . . 41
      3.2.10. Изменение локали браузера . . . 42
      3.2.11. Изменение часового пояса . . . 42
      3.2.12. Использование внешних хостов . . . 44
      3.2.13. Использование прокси-серверов . . . 45
   3.3. Работа с Cypress . . . 46
      3.3.1. Выбор браузера . . . 47
      3.3.2. Выбор определенной версии Cypress . . . 47

3.3.3. Запись видео . . . 47
3.3.4. Дополнительная функциональность . . . 48
3.3.5. Запись выполненных тестов в Cypress Dashboard . . . 50
3.4. Работа с Playwright . . . 51
3.4.1. Выбор браузера . . . 51
3.4.2. Запись видео . . . 52
3.4.3. Дополнительная функциональность . . . 52
3.4.4. Дополнительная функциональность . . . 53
3.5. Использование инструментов разработчика в Chrome . . . 56
3.5.1. Запуск нужного браузера . . . 56
3.5.2. Запись видео . . . 57
3.5.3. Дополнительная функциональность . . . 57
4. Конфигурация . . . 59
4.1. Лицензионный ключ . . . 59
4.1.1. Вывод списка лицензионных ключей . . . 60
4.1.2. Обновление лицензионного ключа . . . 61
4.1.3. Множество лицензионных ключей . . . 61
4.1.4. Удаление лицензионного ключа . . . 62
4.1.5. Срок действия лицензионного ключа . . . 63
4.1.6. Обновление лицензионного ключа из внешнего секрета . . . 64
4.2. Пользователи и квоты . . . 64
4.2.1. Пользователи . . . 64
4.2.2. Квоты . . . 72
4.2.3. Конфигурационный объект . . . 73
4.2.4. Набор браузеров . . . 76
4.2.5. Набор устройств . . . 87
4.3. Запись видео . . . 88
4.3.1. Настройка S3 хранилища . . . 88
4.3.2. Включение записи видео . . . 90
4.4. Автоматическое обновление версий браузеров . . . 95
4.4.1. Установка . . . 95
4.4.2. Возможности конфигурации . . . 96
4.5. Использование собственного реестра контейнеров . . . 97
4.6. Настройка таймаутов . . . 98
4.6.1. Настройка таймаутов в Moon . . . 99
4.6.2. Настройка других таймаутов . . . 100
4.7. Настройка потребления ресурсов . . . 101
4.7.1. Потребление ресурсов браузерами . . . 101
4.7.2. Потребление ресурсов сервисными образами . . . 101
4.7.3. Качество обслуживания подов . . . 102
4.8. Использование дополнительных доверенных сертификатов TLS . . . 103

4.9. Расширенные настройки . . . 105
4.9.1. Добавление пользовательских аннотаций Kubernetes . . . 106
4.9.2. Добавление пользовательских меток Kubernetes . . . 106
4.9.3. Добавление сетевых политик . . . 106
4.9.4. Использование node selector . . . 107
4.9.5. Использование аффинити . . . 107
4.9.6. Использование Tolerations . . . 107
4.9.7. Запуск браузеров в привилегированном режиме . . . 107
4.9.8. Настройка пользовательского идентификатора пользователя (uid) и группы (gid) для браузерных подов . . . 107
4.9.9. Настройка service account для подов . . . 108
4.9.10. Настройка security context для подов . . . 108
4.10. Обновление версии Moon . . . 109
4.11. Мониторинг . . . 110
4.11.1. Установка . . . 110
4.11.2. Встроенные метрики Moon . . . 111
4.11.3. Фильтрация подов по меткам . . . 111
4.12. Логи . . . 112
4.13. Флаги командной строки (CLI) . . . 112
4.13.1. Флаги контейнера Moon . . . 112
4.13.2. Флаги контейнера Moon Auth . . . 113
4.13.3. Флаги контейнера Moon Basic Auth . . . 114

1. О документе

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

2. Начало работы

2.1. Начало работы

Данный раздел описывает установку Moon с ограничением в четыре параллельных браузерных сессии. Подробная информация о том как установить лицензионный ключ, позволяющий снять это ограничение описан в разделе установка лиценионного ключа.

2.1.1. Установка в Kubernetes

Системные требования

1. Работающий кластер Kubernetes

2. Установленная утилита `kubectl` с настроенным доступом к кластеру
3. Если вы запускаете кластер Kubernetes на виртуальных машинах мы рекомендуем создавать виртуальные машины с насколько возможно большим количеством процессоров. Это позволит избежать проблем с фрагментацией памяти и нехваткой ресурсов. К примеру, если у вас имеется 24 процессора мы рекомендуем создать 3 виртуальные машины с 8 процессорами, а не 12 виртуальных машин по 2 процессора.
4. Если вы запускаете Moon в кластере, развернутом на рабочей станции при помощи инструмента minikube - ознакомьтесь с разделом Вариант 3: у вас Minikube.

Вариант 1: Установка с помощью Helm

Установка с помощью Helm является рекомендуемым способом установки Moon. Мы предполагаем, что используется версия Helm 3. Более старые версии не поддерживаются.

Мы предоставляем готовые Helm чарты, поэтому установка Moon с помощью Helm делается очень просто:

1. Добавьте репозиторий Aerokube, содержащий Helm чарты:

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

2. Для уточнения доступных версий используйте команду:

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

3. Создайте неймспейс:

```
$ kubectl create namespace moon
```

4. Установите или обновите Moon командой:

```
$ helm upgrade --install -n moon moon aerokube/moon2
```

5. Helm чарт для Moon содержит различные параметры конфигурации, которые можно посмотреть командой:

```
$ helm show values aerokube/moon2
```

Для изменения каких-либо параметров используйте аргумент `--set`:

```
$ helm upgrade --install --set=moon.enabled.resources=false -n moon moon
```

```
aerokube/moon2
```

6. По умолчанию, развернутому объекту Ingress присваивается имя хоста `moon.aerokube.local`. Вы можете его изменить командой:

```
$ helm upgrade --install -n moon moon aerokube/moon2 --set
ingress.host=moon.example.com
```

Откройте ссылку http://moon.example.com/ в браузере и вы увидите интерфейс пользователя. В коде Selenium тестов используйте ссылку http://moon.example.com/wd/hub.

7. По умолчанию Moon запускается в режиме HTTP-only. Для включения TLS шифрования, то есть HTTPS, вам необходимо предоставить TLS сертификат и приватный ключ:

```
$ helm upgrade --install -n moon moon aerokube/moon2 --set
ingress.host=moon.example.com --set-file ingress.tlsCert=server.crt --set-file
ingress.tlsKey=server.key
```

Обычно сертификат и приватный ключ предоставляются либо сторонними поставщиками (центрами сертификации) либо отделом безопасности в вашей организации. Для создания тестовой пары сертификат+приватный ключ используйте следующие команды:

```
# Создание CA ключа и сертификата
$ openssl req -x509 -sha256 -newkey rsa:4096 -keyout ca.key -out ca.crt -days 356
-nodes -subj '/CN=My Cert Authority'
# Создание серверного ключа, создание сертификата и подписание сертификата
$ openssl req -new -newkey rsa:4096 -keyout server.key -out server.csr -nodes -subj
'/CN=moon.aerokube.local'
$ openssl x509 -req -sha256 -days 365 -in server.csr -CA ca.crt -CAkey ca.key
-set_serial 01 -out server.crt
```

При использовании таких тестовых сертификатов вам понадобится явно, вручную разрешить браузеру открывать Moon интерфейс.

8. Если Moon устанавливается на машину с архитектурой процессора ARM64 (например, Mac M1 и подобные CPU или облачные ноды Kubernetes с архитектурой ARM64), то выбор доступных браузеров ограничен. Selenium будет работать с Chromium, Firefox или Safari (другие браузеры не имеют версий, совместимых с Linux ARM64). Playwright, Cypress и Puppeteer не будут работать вообще. Для того, чтобы использовать браузеры, совместимые с ARM64, вам необходимо настроить файл `values.yaml` следующим образом:

Использование образов с браузерами, совместимых с ARM64

```
browsers:
```

```
  default:
    playwright: {}
    cypress: {}
    devtools: {}
    selenium:
      MicrosoftEdge: null
      opera: null
      chrome:
        default: 124.0.6367.60-1
        repository: quay.io/browser/chromium
      firefox:
        default: 125.0.3-1
        repository: quay.io/browser/firefox
      safari:
        default: 613.1.6.1
        repository: quay.io/browser/webkit
```

Вариант 2: Установка с помощью Minikube

Каждый браузер по умолчанию требует 1 процессор и 2 гигабайта памяти. В вашем кластере Minikube мы рекомендуем как минимум 4 процессора и 8 гигабайт памяти. При меньшем количестве процессоров поды могут не стартовать из-за нехватки вычислительных ресурсов. Мы не рекомендуем использовать Docker драйвер для Minikube.

1. Запуск Minikube на Linux

```
$ minikube start --cpus=4 --memory=8G --disk-size=20G --driver kvm2
```

Запуск Minikube на MacOS и процессором архитектуры x86

```
$ minikube start --cpus=4 --memory=8G --disk-size=20G --driver=hyperkit
```

Запуск Minikube на MacOS и процессором архитектуры ARM64 (M1 и подобные процессоры)

```
$ brew install qemu
$ brew install socket_vmnet
$ brew tap homebrew/services
$ HOMEBREW=$(which brew) && sudo ${HOMEBREW} services start
socket_vmnet
$ minikube start --cpus=4 --memory=8G --disk-size=20G --driver qemu
--network socket_vmnet
```

Запуск Minikube на Windows

```
$ DISM /Online /Enable-Feature /All /FeatureName:Microsoft-Hyper-V #
Enable Hyper-V
```

```
$ minikube start --cpus=4 --memory=8G --disk-size=20G --driver=hyperv
```

1. Включите поддержку Ingress в Minikube:

   ```
   $ minikube addons enable ingress
   ```

   Данная команда может не работать на некоторых версиях Minikube для Mac M1 и подобных процессоров.

2. Сам процесс установки выполняется с помощью Helm и был описан выше.
3. Настройка доступа в Moon по сети:
   a. **Вариант 1.** Используйте команду `minikube ip` для обновления настроек Moon.
      i. Обновление с помощью вывода команды `minikube ip`:

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

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

      ii. Добавьте `moon.aerokube.local` в файл `/etc/hosts`:

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

      На Windows вам может понадобиться обновить файл вручную.

   b. **Вариант 2.** Используйте туннель minikube. Этот вариант возможен только при использовании minikube с драйвером Docker.
      i. Добавьте `moon.aerokube.local` в `/etc/hosts`:

      ```
      $ sudo echo '127.0.0.1 moon.aerokube.local' >> /etc/hosts
      ```

      ii. Запустите туннель Minikube в отдельном терминале, введите пароль, когда потребуется:

      ```
      $ minikube tunnel
      ```

4. Откройте ссылку http://moon.example.com/ в браузере, вы увидите интерфейс пользователя. В коде Selenium тестов используйте ссылку http://moon.example.com/wd/hub.

2.1.2. Установка в Openshift

1. Системные требования:
   - Работающий кластер Openshift версии 4.x
   - Установленный `oc` клиент с настроенным доступом к кластеру. Установка была протестирована на пользователе с правами администратора кластера Openshift.
2. Создайте проект для Moon (аналог неймспейса в Kubernetes):

   ```
   $ oc new-project moon
   ```

   В следующих шагах мы предполагаем, что созданный проект называется `moon`.

3. Добавьте репозиторий с Helm чартами:

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

4. Установите или обновите Moon командой:

   ```
   $ helm upgrade --install --set ingress.openshift=true -n moon moon aerokube/moon2
   ```

   Флаг `-n moon` указывает на проект, созданный на предыдущем этапе.

5. Отредактируйте идентификаторы пользователя и группы конфигурация объекта для совпадения со значениями, разрешенными политиками Openshift (например, установите идентификатор `1000650000`, конкретное значение зависит от конфигурации Openshift):

   ```
   $ oc edit config.moon.aerokube.com default -n moon
   ```

Для тестирования на локальной машине вы можете использовать Openshift Local. В этом случае вам необходимо дополнительно передать имя хоста для Ingress следующим образом:

```
$ helm upgrade --install --set ingress.openshift=true --set
ingress.host=moon.apps-crc.testing -n moon moon aerokube/moon2
```

После запуска подов Moon добавьте `moon.apps-crc.testing` в `/etc/hosts`:

```
$ sudo echo '127.0.0.1 moon.apps-crc.testing' >> /etc/hosts
```

2.1.3. Рекомендации по Работе с Helm

Values-файлы Helm

Как было сказано выше, использование Helm чарта является рекомендуемым способом установки Moon в Kubernetes и Openshift. Наш Helm чарт поставляется с рекомендуемыми значениями настроек по-умолчанию, которые будут работать "из коробки" в большинстве случаев. Тем не менее, бывают случаи, когда необходимо установить Moon с заранее включенными продвинутыми функциями и иметь возможность легко воспроизводить данную конфигурацию позднее. Например, по-умолчанию в Moon отключена видеозапись браузерных сессий, но она может требоваться в ваших сценариях тестирования. Аналогично, вы можете захотеть запускать браузеры в нескольких неймспейсах или использовать дополнительные доверенные TLS сертификаты.

Хотя все эти возможность могут быть включены ручным редактированием соответствующих объектов Moon (конфигурационного объекта, набора браузеров, набора устройств) с помощью команды `kubectl edit`, все эти изменения будут перезаписаны Helm при следующей выкладке Helm чарта. Helm предоставляет лучший способ для хранения воспроизводимой конфигурации выкладки - values-файлы. Это работает следующим образом:

1. Каждый Helm чарт содержит файл `values.yaml` с рекомендуемыми значениями параметров по-умолчанию. Этот файл распространяется вместе с другими частями чарта. Например, актуальный файл `values.yaml` для Moon 2.x хранится здесь.
2. Если требуется поменять некоторые значения по-умолчанию на пользовательские значения, вы создаете локальный файл `values.yaml`, содержащий только значения, которые вы хотите переопределить. Например, в Helm чарте Moon число реплик Moon может быть задано, используя поле `deployment.replicas` в `values.yaml`. При этом ваш файл `values.yaml` будет выглядеть так:

   ```yaml
   deployment:
     replicas: 3
   ```

   Такой файл следует хранить под предпочитаемой вами системой контроля версий. Такой подход гарантирует, что вы никогда не потеряете пользовательские настройки для установки Moon.

3. Для того, чтобы применить свой файл `values.yaml` при установке (переустановке) Moon - передайте полный путь до `values.yaml` в команде Helm:

   ```
   $ helm upgrade --install -f путь/до/values.yaml -n moon moon aerokube/moon2
   ```

   Переопределение параметров `values.yaml` аналогично, но более удобно, чем передача тех же самых ключей через флаг `--set`, например:

   ```
   $ helm upgrade --install --set deployment.replicas=3 -n moon moon aerokube/moon2
   ```

Настройка Пользовательских Ресурсов Moon

В Moon 2.x все настройка хранятся как встроенные объекты Kubernetes, называемые пользовательскими ресурсами:

- Квоты - хранят конфигурацию для одного пользователя Moon.
- Конфигурационные объекты - хранят различные конфигурационные опции.
- Наборы браузеров - хранят настройки запуска браузеров.
- Наборы устройств - предоставляют информацию о списке доступных для мобильной эмуляции мобильных устройств.
- Лицензии - хранят установленные лицензионные ключи Moon.

Наш Helm чарт имеет отдельный ключ в `values.yaml`, соответствующий каждому из этих ресурсов:

Пользовательские ресурсы Moon могут быть также настроены через файл values.yaml

```
quota: # Хранит список квот
  quota1:
    # Параметры квоты Quota1
  quota2:
    # Параметры квоты Quota2

configs: # Хранит список конфигурационных объектов
  config1:
    # Параметры конфигурационного объекта Config1
  config2:
    # Параметры конфигурационного объекта Config2

browsers: # Хранит список наборов браузеров
  browsersset1:
    # Параметры набора браузера Browsersset1
  browsersset2:
    # Параметры набора браузера Browsersset2

license: # Хранит пользовательский лицензионный ключ для Moon (в данный момент Helm
чарт поддерживает только один лицензионный ключ)
```

Как вы видите, для всех пользовательских ресурсов кроме лицензий вы можете определить несколько разных объектов: несколько конфигурационных объектов, несколько квот и несколько наборов браузеров. Все поля, которые вы выставляете в спецификации пользовательского ресурса преобразуются "как есть" в соответствующий ресурс Kubernetes. Например, пусть в `values.yaml` определен набор браузеров:

Пример определения набора браузеров в values.yaml

```
browsers:
  my-browserset:
    selenium:
```

```
      firefox:
        repository: quay.io/browser/firefox
      chrome:
        repository: quay.io/browser/chrome
```

Когда вы применяете Helm чарт с таким определением в файле values.yaml, будет автоматически создан такой ресурс Kubernetes типа BrowserSet (т.е. набор браузеров) Kubernetes:

Эквивалентные объект типа "набор браузеров" в Kubernetes

```yaml
apiVersion: moon.aerokube.com/v1
kind: BrowserSet
metadata:
  name: my-browserset
  namespace: moon # Здесь будет имя неймспейса, куда установлен Moon
spec:
  selenium:
    firefox:
      repository: quay.io/browser/firefox
    chrome:
      repository: quay.io/browser/chrome
```

Если вы создаете несколько объектов в списке наборов браузеров в `values.yaml`, то будет создано соответствующее количество пользовательских ресурсов Kubernetes.

2.2. Архитектура

2.2.1. Компоненты Moon

Компоненты Moon

[moon components]

Кластер Moon состоит из нескольких компонентов:

1. Одна или несколько реплик приложений `Moon`. Их основная задача заключается в старте/остановке контейнеров с браузерами. Реплики обычно доступны как сервисы Kubernetes на стандартном порту Selenium - `4444`. Все тесты должны отправлять запросы в этот сервис Kubernetes. Это же приложение предоставляет API, используемый для получения информации о запущенных браузерах (в версии Moon 1.x это отдельное приложение под названием `Moon API`).
2. Одна или несколько реплик приложений `Moon Conf`. Это приложение перезапускает поды Moon, когда вы обновляете лицензионный ключ.
3. Одна или несколько реплик приложений `Moon UI`. `Moon UI` собирает информацию от Moon и визуализирует ее. Реплика обычно доступна на стандартном HTTP порту `8080`.
4. Запущенные поды с браузерами.

2.2.2. Режимы работы Moon

В Moon 2.x возможно два режима запуска и работы: **используя один неймспейс** и **используя множество неймспейсов**.

Режим одного неймспейса

[single-namespace-mode]

В режиме одного неймспейса сам Moon и все запущенные браузерные поды выполняются в одном неймспейсе Kubernetes. В версии Moon 1.x это был единственный возможный режим. Он может использоваться в ситуации когда с Moon работает только одна команда или в ситуации когда вам не нужно ограничивать количество используемых браузеров. По умолчанию Moon запускается именно в этом режиме.

Режим множества неймспейсов

[multiple-namespaces-mode]

В режиме множества неймспейсов Moon запускается в одном неймспейсе, а браузеры запускаются в другом, отдельном неймспейсе. Общее количество таких неймспейсов не ограничено. Этот режим нужен когда вам необходимо контролировать вычислительные ресурсы, браузеры и правила сетевого доступа (network policies) для каждой команды по-отдельности. Настройка Moon для работы в данном режиме описана здесь.

2.2.3. Содержимое браузерных подов

В дополнение к контейнеру с браузером каждый под созданный в Moon содержит один или более образов сервисных контейнеров.

Table 1. Служебные контейнеры

Название	Назначение	Условия, при которых запускается
ca-certs	Выпуск сертификатов для браузера	Всегда при инициализации пода
defender	Создание и ограничение в одну браузерную сессию в поде, обработка таймаутов	Всегда
proxy	Авторизация в прокси для Selenium	Когда используется прокси
video-recoder	Запись экранных видео работающих браузеров	Когда сконфигурирована запись видео
vnc-server	Предоставление VNC доступа к сессиям	Если отображается окно браузера
x-server	Предоставление X сервера для запуска браузеров в экранном режиме	Если отображается окно браузера

2.3. Рекомендуемые настройки для кластера

- **Используйте как можно большие размеры нод кластера**. К примеру, имея всего 100 процессоров лучше запустить 5 нод с 20ю процессорами на каждой, чем 50 нод с двумя процессорами в каждом. Браузерные поды в некоторых случаях требуют более двух процессоров и в случае нехватки вычислительных ресурсов вы столкнетесь с постоянной фрагментацией кластера.
- **По возможности не создавайте кластерных нод с операционными системами RedHat \ CentOS.** Из-за специфических настроек безопасности (фаервол, SeLinux) эти дистрибутивы сложны в конфигурировании и вы можете столкнуться с непонятными проблемами.
- **Рекомендуемый сетевой интерфейс контейнера - Calico. По возможности не используйте Flannel**. У Calico производительность выше, чем у Flannel, особенно в больших кластерах.
- **Не стремитесь ограничиваться одной репликой Kubernetes API**. Moon использует Kubernetes API для запуска и удаления подов с браузерами. Если вы планируете запускать сотни параллельных сессий браузера, наблюдайте за системными метриками Kubernetes API (Kubernetes master). Перегруженное Kubernetes API может перестать отвечать на запросы, что приведет к зависанию подов с браузерами.

2.4. Необходимые права доступа

Moon не требует сложных прав доступа и ему вполне достаточно настроек доступа Kubernetes по умолчанию. По умолчанию Moon запускает браузеры в том же неймспейсе `moon`, где он запущен (**режим одного неймспейса**). Версия Moon 2.0.0 и выше поддерживает множество неймспейсов Kubernetes. Это позволяет вам запускать Moon в неймспейсе `moon`, а браузеры в произвольном количестве неймспейсов для разных пользователей (**режим множества неймспейсов**). Это дает возможность легко задать максимальное количество браузеров, доступных каждой команде.

2.4.1. Режим одного неймспейса

Следующая таблица содержит права доступа, необходимые для запуска Moon в режиме одного неймспейса:

Table 2. Необходимые права доступа в режиме одного неймспейса

Право	Назначение
Операции **get**, **watch**, **list**, **create**, **delete**, **update** и **patch** для подов	Управление браузерными подами
Операции **get**, **watch**, **list**, **create**, **delete**, **update** и **patch** для config maps	Доступ пользователей и групп к подам
Операции **get**, **watch**, **list**, **create**, **delete**, **update** и **patch** для deployments и replica sets	Управление лицензией

Операции **get**, **watch** и **list** для ресурсов Moon в API группе `moon.aerokube.com`	Конфигурация Moon. Лицензия Moon (`licenses.moon.aerokube.com`) распространяется на весь кластер, таким образом ClusterRole является необходимым ресурсом.

2.4.2. Режим множества неймспейсов

В режиме множества неймспейсов необходимые права доступа отличаются:

Table 3. Необходимые права доступа в неймспейсе, где запущен Moon

Право	Назначение
Операции **get**, **watch** и **list** - сбор информации о неймспейсе	Контроль количества браузеров запущенных в одном неймспейсе
Операции **get**, **watch** и **list** для подов	Анализ подов в неймспейсе Moon
Операции **get**, **watch**, **list**, **create**, **delete**, **update** и **patch** для deployments и replica sets	Управление лицензией
Операции **get**, **watch** и **list** ресурсов Moon в API группе `moon.aerokube.com`	Конфигурация Moon. Лицензия Moon (`licenses.moon.aerokube.com`) распространяется на весь кластер, таким образом ClusterRole является необходимым ресурсом.

Для каждого пользовательского неймспейса Moon требует следующих прав:

Table 4. Необходимые права для пользовательского неймспейса

Право	Назначение
Операции **get**, **watch**, **list**, **create**, **delete**, **update** и **patch** для подов	Управление браузерными подами
Операции **get**, **watch**, **list**, **create**, **delete**, **update** и **patch** для config maps	Управление доступом пользователей к браузерным подам

2.5. Различие версий Moon 2.x и Moon 1.x

Версия Moon 2.x - новая версия Moon, включающая множество нововведений и улучшений. Данный раздел дает представление о наиболее значимых изменениях.

- **Множество неймспейсов Kubernetes**. Moon версии 1.x позволяет запускать все браузеры в **общем** (одном) неймспейсе Kubernetes. Однако, общий кластер Moon часто используется разными командами в компании. Часто необходимо уметь ограничить количество используемых браузеров для каждой команды. Ограничение количества запускаемых браузеров фактически означает ограничение вычислительных ресурсов для каждой команды. В Kubernetes эта проблема уже решена с помощью так называемых **неймспейсов**. Неймспейсы можно рассматривать как проекты, которым администратор Kubernetes выделил некоторое количество вычислительных ресурсов. В Moon версии 2.x вы можете создать неограниченное количество различных неймспейсов, по одному на каждую команду? а затем сконфигурировать Moon, чтобы запускать браузеры в этих

неймспейсах. Это дает администратору Kubernetes полный контроль над вычислительными ресурсами для каждой команды. Действие лицензионного ключа Moon распространяется на все эти неймспейсы. В Moon 1.x отдельный неймспейс для команд требовал отдельной копии Moon и отдельного лицензионного ключа. В некоторых случаях это ограничивало возможность запуска большего количества браузеров в период высокой нагрузки. Во Moon 2.x действие одного лицензионного ключа распространяется на множество неймспейсов и, таким образом, если емкость лицензионного ключа позволяет - каждая команда при необходимости может получить больше браузерных сессий чем обычно. Более подробно это описано в разделе ref:#architecture[Архитектура].

- **Улучшенная конфигурация**. Версия Moon 1.x использует конфигурационные файлы в формате JSON. Например, файл с описанием браузеров в версии Moon 1.x хранится в Kubernetes и выглядит так:

Стандартный файл с описанием браузеров версии Moon 1.x

```
{
  "firefox": {
    "default": "95.0",
    "versions": {
      "95.0": {
        "image": "browsers/firefox:95.0",
        "port": "4444",
        "path": "/wd/hub"
      }
    }
  }
}
```

В версии Moon 2.x описание браузеров является встроенной функциональностью Kubernetes, которая называется **browser set** (объект конфигурации браузеров) и выглядит так:

Описание конфигурации браузеров в Moon 2.x

```
apiVersion: moon.aerokube.com/v1
kind: BrowserSet
metadata:
  name: default
  namespace: moon
spec:
  selenium:
    firefox:
      repository: quay.io/browser/firefox
    chrome:
      repository: quay.io/browser/chrome
```

Вы можете простым способом смотреть и изменять эти настройки с помощью любого

клиента Kubernetes, например:

```
$ kubectl get browsersets -n moon -o yaml # Show all available browser sets
$ kubectl edit browserset default -n moon
```

То же самое касается других настроек. Даже работа с лицензионным ключом Moon возможна с помощью стандартных инструментов:

```
$ kubectl get license -n moon
NAME      LICENSEE          SESSIONS   EXPIRES
default   Acme Inc.         10         2022-10-11T18:38:42Z
```

Каждое изменение в этих объектах автоматически проверяется Kubernetes перед сохранением, что предотвращает возможные ошибки.

- **Новые версии браузеров доступны автоматически**. В Moon версии 1.x требовалось вручную добавлять образ для каждой версии браузера в файл со списком версий. Если версия браузера отсутствовала в списке, Moon 1.x не мог запустить браузер. Единственное, что требуется для Moon версии 2.x это указать образ репозитория для каждого типа браузера. После этого новые версии определяются автоматически.
- **Улучшенная производительность браузеров**. Moon версии 2.x использует совершенно новую архитектуру старта браузеров, которая запускает только необходимые компоненты операционной для работы конкретного браузера. Например, компоненты, управляющие элементами окон браузера, используются при старте только если браузер запущен в экранном режиме. Это дает возможность сделать образы более легковесными, с более быстрой загрузкой и более быстрой отработкой команд.
- **Меньшее использование облачных вычислительных ресурсов**. Переработанная архитектура старта браузеров позволяет как минимум на 20% уменьшить использование облачных ресурсов (процессоров, памяти, сетевого трафика).
- **Улучшенное сетевое взаимодействие**. Moon версии 1.x для коммуникации с браузерными подами во многом полагается на реализацию DNS от Kubernetes (например e.g. CoreDNS). Известно что сервис DNS часто подвержен проблемам с кэшированием и специфичными для облака сетевыми ошибками, что иногда приводит к разрыву браузерной сессии. Moon версии 2.x полагается на IP адрес пода и не зависит от сервиса DNS вообще.
- **Отсутствие встроенных механизмов аутентификации**. Moon версии 1.x поддерживает только базовую HTTP аутентификацию. Moon версии 2.x в свою очередь не предоставляет встроенного механизма аутентификации вообще. Вместо этого для реализации любого нужного вам механизма аутентификации (например mutual TLS authentication) вы можете использовать уже существующее программное обеспечение совместимое с Kubernetes, например Nginx Ingress Controller). Moon получает имя пользователя из HTTP заголовков `Authorization` или `X-Moon-Quota`. Более развернуто это описано в раздел Пользователи.
- **Поддержка OpenID Connect**. Moon версии 2.x содержит готовый к использованию

контейнер для использования механизма аутентификации OpenID Connect. Это позволяет легко использовать уже существующий публичную или корпоративную реализацию протокола OAuth. Например, вы можете легко загрузить существующих Github пользователей.

- **Улучшена поддержка самоподписанных корневых центров сертификации TLS**. Компании часто используют самоподписанные TLS сертификаты для внутренних веб-сервисов. В Moon 2.x настройка работы с такими самоподписанными сертификатами делается один раз в конфигурации Moon для всех компонентов Moon и версий браузеров (нужно прописать сертификат внутреннего центра сертификации).

  Moon 2 самоподписанный корневой сертификат

  ```
  apiVersion: moon.aerokube.com/v1
    kind: Config
    metadata:
      annotations:
      name: default
      namespace: moon
    spec:
      additionalTrustedCAs: |
        -----BEGIN CERTIFICATE-----
        ...
  ```

- **Расширенные возможности Selenium**. Moon полностью совместим с W3C Webdriver protocol. Это означает что все возможности Selenium 4.x будут доступны сразу после установки. В дополнение к стандартной функциональности Moon дает возможность расширения некоторых возможностей, таких как взаимодействие с буфером обмена или сохранение файлов из контейнера. Например, вы легко можете скопировать и вставить произвольные текстовые данные и изображения из ваших тестов в буфер обмена браузера. Также вы можете ходить через прокси-сервис с аутентификацией.

2.6. Moon и другие решения

Moon использует лучшие практики и возможности существующих решений по автоматизации браузерного тестирования и добавляет еще больше полезного:

1. **Универсальный инструмент для тестирования в браузерах**. Moon поддерживает все самые популярные технологии тестирования в браузерах, такие как Selenium, Playwright, Cypress, Puppeteer без необходимости какой-либо дополнительной настройки. Новые образы с браузерами собираются и публикуются автоматически.
2. **Автоматическая неограниченная масштабируемость.** В кластере у вас будет всегда достаточное количество браузеров любой желаемой версии. При поднятии кластера в облачных платформах типа Yandex Cloud или Cloud.ru вы можете настроить автоматическое масштабирование, которое будет применяться в зависимости от текущей загрузки. Это позволит совместить производительность и приемлемую цену вычислительных ресурсов.
3. **Не имеет внутреннего состояния (stateless)**. Selenoid и Selenium Grid 3.x хранят в

памяти всю информацию о запущенных в данный момент браузерах. Selenium Grid 4.x с той же целью использует механизм хранения списка сессий в key-value хранилище (например Redis). Если по какой-то причине информация о запущенных сессиях браузера теряется - все сессии закрываются. Moon в свою очередь не хранит состояние сессии в себе (это делается в Kubernetes), может реплицироваться на несколько датацентров. При рестарте реплики или ее остановке сессии не обрываются.

4. **Полный контроль потребления вычислительных ресурсов**. Moon позволяет легко настроить потребляемые вычислительные ресурсы для каждого компонента. Расход вычислительных ресурсов становится предсказуемым, стоимость кластера становится легко подсчитать.
5. **Высокая доступность**. Любое изменение настроек кластера не прерывает работу браузерных сессий. Каждый компонент кластера завершает работу безопасно без разрыва сессий.

3. Основная функциональность

3.1. Веб интерфейс

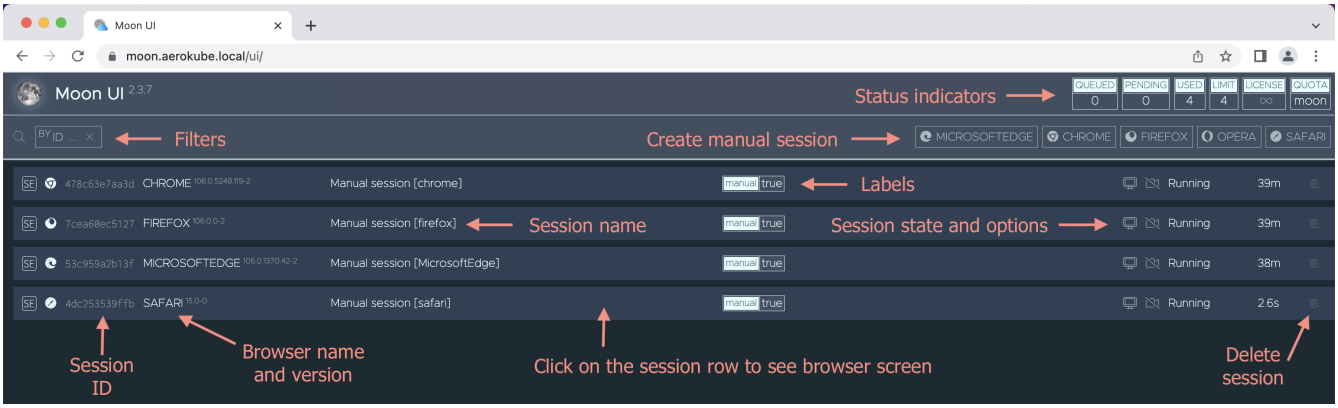
1. Данный веб-интерфейс доступен начиная с версии Moon 2.4.0.
2. Данный веб-интерфейс включен по-умолчанию, начиная с версии Moon 2.6.2. Для более старых версий его поддержку нужно включить явно при установке Moon. Для включения - используйте следующий параметр в значениях Helm чарта:

```
deployment:
  experimentalUI: true
```

После включения новый веб интерфейс автоматически доступен по обычному адресу Moon с добавлением `/ui/` в конец, например `https://moon.example.com/ui/`. Старый интерфейс по прежнему доступен по старому адресу (например `https://moon.example.com/`), в дальнейшем, планируется отказаться от старого интерфейса полностью.

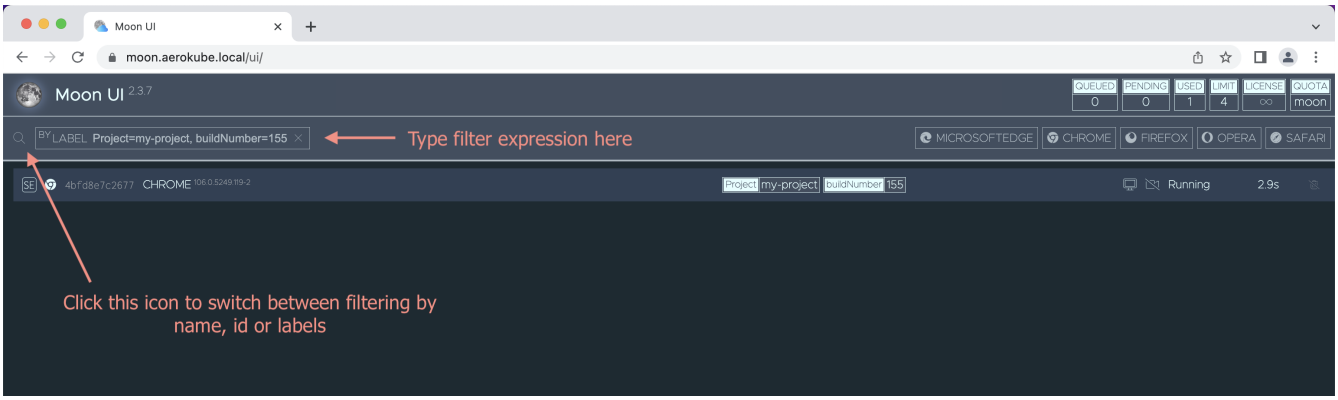
В Moon реализован пользовательский интерфейс, который позволяет выводить список браузерных сессий, фильтровать их по какому-либо признаку, отображать экран браузера, запускать сессии для ручного тестирования и так далее.

Общий вид веб интерфейса Moon UI



Главная страница веб интерфейса отображает список стартующих и уже запущенных браузерных сессий. Для каждой сессии вы можете увидеть имя браузера, версию, название теста, метку, статус, длительность работы и так далее. Для удаления текущей сессии вам необходимо **дважды кликнуть** на кнопку с иконкой мусорной корзины (два клика необходимы для предотвращения случайного нажатия и удаления сессии). Для того чтобы создать сессию для **ручного тестирования**, нажмите на кнопку с названием браузера вверху веб интерфейса.

Фильтрация в интерфейсе Moon UI



В одном кластере Moon параллельно могут запускаться десятки, сотни или даже тысячи браузерных сессий. **Используйте фильтрацию**, чтобы найти сессию, относящуюся к вашему проекту или определенному номеру сборки. Вы можете отфильтровать сессии по **id**, **имени** and **меткам Kubernetes**:

- **Идентификатор** сессии это уникальное значение, которое Moon автоматически генерирует для каждой сессии. Изменить его нельзя. Полный идентификатор выглядит так (пример): `chrome-73-0-ac15ffaa-e641-4c7f-a54c-f25b5be1f135`. В веб интерфейсе отображается только несколько первых символов из этого длинного значения.
- **Название** сессии - это задаваемое в произвольной форме значение для того, чтобы, при необходимости, описать цель создания сессии. Обычно название совпадает с названием проверяемого тест кейса. Для изменения названия сессии используйте капабилити **name** для Selenium либо URL параметр **name** для Playwright, Cypress или Developer tools, например:

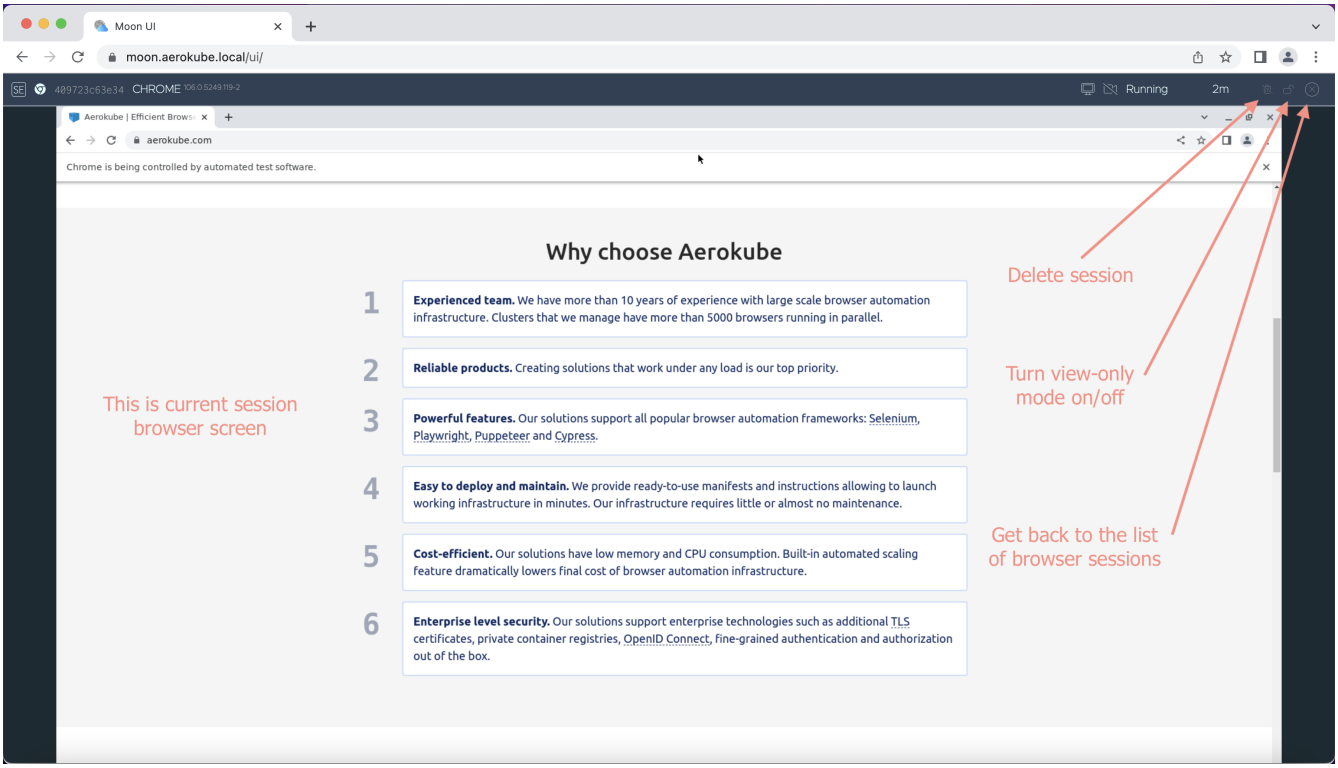
```
wss://moon.example.com/playwright/chromium?name=MyTestCaseName
```

- **Метки** сессии это пары ключ-значение, задаваемые в произвольной форме, позволяющие вам добавить дополнительные метаданные к каждой сессии. Это может быть, к примеру, номер сборки, имя проекта, информация о номере релиза и так далее. Для добавления меток используйте функциональность **labels capability** в Selenium или задайте их в **конфигурации браузеров**. Каждая метка в Moon конвертируется в Kubernetes label и добавляется на соответствующий под. При задании меток используются точно такой же синтаксиси, как в Kubernetes label selectors. Например, если заданы метки `project` (название проекта) и `buildNumber` (номер сборки), то по ним можно фильтровать так:

  Фильтрация по меткам

  ```
  project=MyCoolProject # Полное совпадение одной из меток

  project=MyCoolProject,buildNumber=42 # Полное совпадение значений меток project и buildNumber

  project in (MyCoolProject, AnotherProject),buildNumber!=42 # Выбрать project из списка

  project notin (MyCoolProject, AnotherProject),!buildNumber # Выбрать project не из списка, buildNumber не задано

  project!=AnotherProject,buildNumber # Любое значение project кроме AnotherProject, buildNumber выставлено
  ```

Веб интерфейс Moon и экран браузера



При нажатии на строчку в списке сессий будет автоматически отображен экран браузера, если это действие применимо к сессии. Например, для сессии, запущенной в **режиме без**

экрана (headless) при нажатии на строчку не произойдет ничего. По умолчанию экран браузера отображается только в режиме для чтения, чтобы случайно не вмешаться в работу уже запущенных тестов. Для того чтобы начать взаимодействие с браузером - нажмите на значок замка (🔒) и экран разблокируется. Кликните на значок еще раз - экран браузера станет вновь "только для чтения". На экране браузера вы можете наблюдать выполнение автоматического теста и вмешиваться при необходимости. При запуске браузера для ручного тестирования вы можете использовать экран браузера для поэтапного прохождения вашего тестового сценария. Все возможности браузера (например, инструменты разработчика) доступны и работают в точности так же как и на физической машине. Используйте привычные сочетания клавиш **Ctrl+C\Ctrl+V** или **Cmd+C\Cmd+V** для копирования и вставки значений с вашего компьютера в экран браузера в Moon.

3.1.1. Консоль

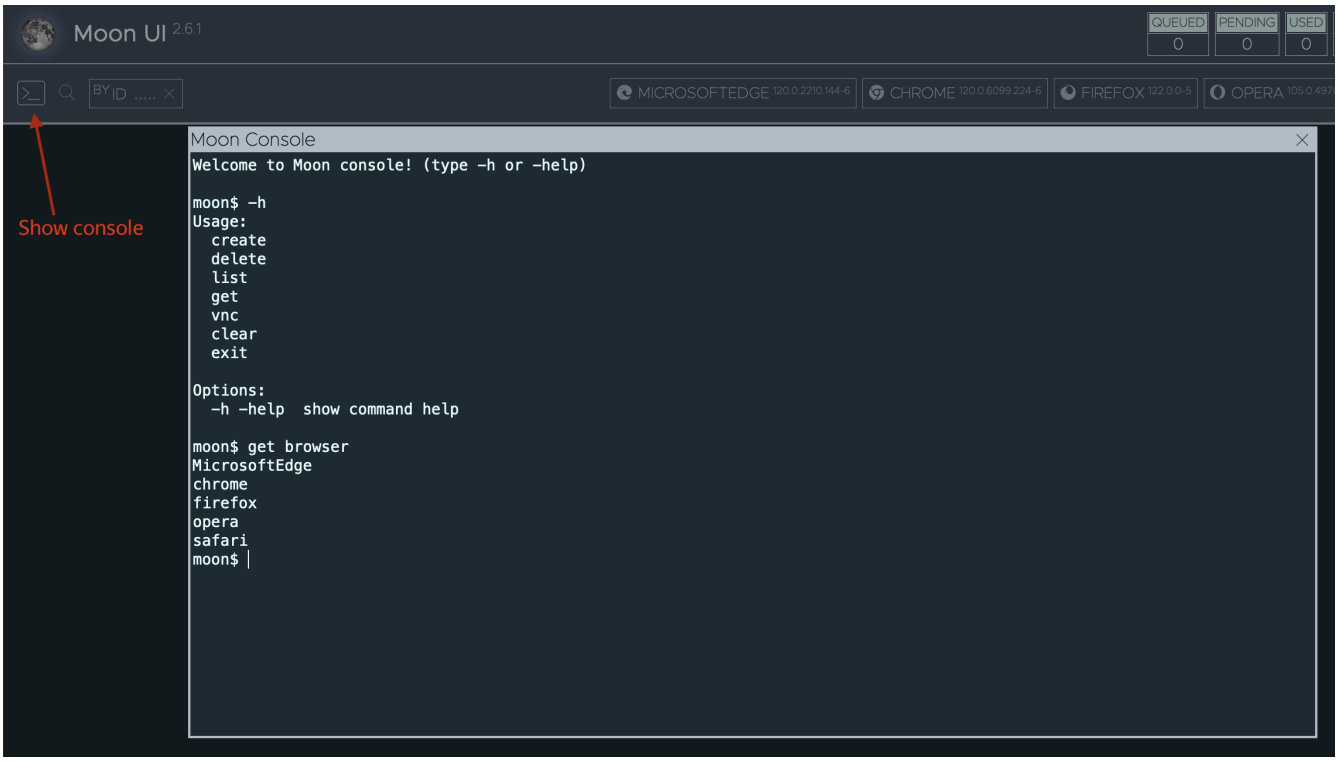
Эта функциональность доступна начиная с версии Moon 2.6.1.

Консоль это интерфейс командной строки доступный в экране пользовательского интерфейса Moon, который дает больше возможностей для ручного тестирования. Имеются следующие возможности:

- Вывод списка доступных браузеров и мобильных устройств эмуляции.
- Запуск, вывод списка, удаление браузерных сессий с работающими браузерами и с включенной эмуляцией мобильных устройств.
- Легкое предоставление произвольных дополнительных возможностей браузера.
- Открытие сессии VNC для ручного тестирования

Консоль выглядит так:

Консоль Moon



Чтобы открыть консоль:

- **Вариант 1.** Кликните на кнопку, показанную на картинке.
- **Вариант 2.** Нажмите на значок `~` (тильда) на клавиатуре.

Консоль Moon работает подобно стандартному терминалу Unix. Для вывода списка доступных команд наберите:

Вывод списка доступных команд

```
moon$ -h
Usage:
  create
  delete
  list
  get
  vnc
  clear
  exit
```

Описание синтаксиса команд доступно при добавлении флага `-h` или `-help`:

Помощь по синтаксису команд

```
moon$ delete -h
Desc:
  delete - stop session

Usage:
  delete <session-id>
```

Предыдущие набранные команды доступны при нажатии стрелок вверх и вниз на клавиатуре. Очистить вывод предыдущих команд можно стандартным способом:

Очистка экрана консоли

```
moon$ clear
```

Для выхода из консоли просто закройте окно кнопкой либо наберите:

Выход из консоли

```
moon$ exit
```

Список доступных браузеров можно вывести так:

Вывод списка доступных браузеров

```
moon$ get browser
```

```
opera
safari
MicrosoftEdge
chrome
firefox
```

Для вывода списка только последних пяти версий браузера `chrome` наберите:

Вывод списка доступных браузеров

```
moon$ get browser -n chrome -l 5
120.0.6099.224-6 (default)
# Если в объекте browser set определено больше версий, они будут показаны здесь
```

Вывод списка доступных мобильных устройств:

Вывод списка доступных мобильных устройств

```
moon$ get device -e "iPhone X" # Частичное совпадение по имени
"Apple iPhone X"
"Apple iPhone XR"
"Apple iPhone Xs"
"Apple iPhone Xs Max"
moon$ get device -e "iPhone X$" # Совпадение по регулярному выражению
"Apple iPhone X"
```

Для запуска браузера Chrome наберите:

Запуск настольного браузера

```
moon$ create browser -n chrome # Версия Chrome по-умолчанию
chrome-120-0-124bcfdf-6f03-424c-80b4-6c2ee5b2f36f # Это ID запущенной сессии
moon$ create browser -n chrome -v 120.0 # Конкретная версия Chrome
moon$ create browser -n chrome -caps '{"goog:chromeOptions": {"args": ["start-
maximized"]}}' # Дополнительные капабилити Selenium в формате JSON
```

Запуск мобильных устройств:

Запуск мобильной эмуляции

```
moon$ create device -n "Apple iPhone Xs" -url https://aerokube.com/
chrome-120-0-6099-224-6-46e1198c-f73d-401d-be6c-6e127ef53f24
```

Открытие VNC для браузерной сессии:

Просмотр браузерной сессии

```
moon$ vnc chrome-120-0-6099-224-6-46e1198c-f73d-401d-be6c-6e127ef53f24
```

Список запущенных сессий браузера:

Список сессий

```
moon$ list
chrome-120-0-124bcfdf-6f03-424c-80b4-6c2ee5b2f36f
```

Удаление работающей сессии:

Удаление браузерной сессии

```
moon$ delete chrome-120-0-124bcfdf-6f03-424c-80b4-6c2ee5b2f36f
```

3.2. Работа с Selenium

Мы создали несколько простых проектов с исходным кодом, демонстрирующих использование Moon с разными Selenium библиотеками:

- Codeception
- Java
- Protractor
- Python
- Webdriver.io

Запуск тестов Selenium в Moon является довольно простой задачей. Используйте следующий Selenium URL в ваших тестах:

Selenium URL

```
https://moon.example.com/wd/hub
```

Moon полностью совместим с W3C WebDriver specification, все стандартные возможности Selenium будут доступны сразу после установки. Выбор браузера делается выставлением capability `browserName` в вашем коде, например:

Пример теста Selenium на Python

```
from selenium import webdriver

capabilities = {
    "browserName": "chrome"
}

driver = webdriver.Remote(
    command_executor='https://moon.example.com/wd/hub',
    desired_capabilities=capabilities
```

```
)
```

Версия браузера, которая будет использована для теста, зависит от конфигурации Moon, по умолчанию берется самая последняя доступная версия. Явно задать версию можно с помощью значения capability `browserVersion`:

Задание конкретной версии браузера

```
capabilities = {
    "browserName": "chrome",
    "browserVersion": "96.0"
}
```

С помощью расширений протокола WebDriver Moon предоставляет дополнительные возможности, описанные ниже.

3.2.1. Дополнительные Selenium capability, поддерживаемые Moon

Moon поддерживает набор расширенных Selenium capabilities. Значения, включающие определенную функциональность Moon, задаются в коде в ключе `moon:options`:

Задание Moon capabilities

```
capabilities = {
    "browserName": "chrome",
    "moon:options": { # Все капабилити, специфичные для Moon, находятся в ключе
moon:options
        "enableVideo": True,
        "screenResolution": "1280x1024"
    }
}
```

В строго типизированных языках типа Java или C# для задания Moon capabilities нужно использовать Map (Dictionary), например:

Задание Moon capabilities в Java

```
capabilities.setCapability("moon:options", Map.of(
    "screenResolution", "1280x1024"
));
```

Разрешение экрана: screenResolution

Moon позволяет изменить разрешение экрана браузера:

Тип: строка, формат: <ширина>x<высота>

```
screenResolution: "1280x1024"
```

Можно изменить глубину цвета:

Тип: строка, формат: <ширина>x<высота>x<глубина-цвета>

```
screenResolution: "1280x1024x24"
```

Эта capability задает разрешение экрана рабочего стола, а не размер окна браузера. В большинстве браузеров размер открываемого окна задается по умолчанию, поэтому размер скриншота может быть меньше чем заданное разрешение экрана. Изменить размер окна можно вручную, либо используя функцию `maximize` в Selenium.

Имя теста: name

Во время отладки часто полезно задать конкретное название для каждого теста. Это можно сделать с помощью следующей capability:

Тип: строка

```
name: "myCoolTestName"
```

Основное применение этой capability - отладка тестов в графическом интерфейсе, она показывает название конкретного теста для каждой сессии.

Запись видео: enableVideo, videoName, videoScreenSize, videoFrameRate, videoCodec, pattern

Запись видео необходимо настроить заранее.

Для включения записи видео для сессии добавьте капабилити:

Тип: boolean

```
enableVideo: true
```

- По умолчанию полученная видеозапись называется `video.mp4`. Задать другое имя можно так:

  Тип: строка

  ```
  videoName: "my-cool-video.mp4"
  ```

  В название файла обязательно добавляйте расширение файла - `mp4`.

- По умолчанию записывается вся картинка в экране. Capability `screenResolution` позволяет изменить высоту и ширину записываемого видео. Также с помощью capability вы можете поменять экранный размер записываемого видео. В случае если заданная

capability `videoScreenSize` меньше чем настоящий размер экрана - видео будет обрезано начиная с верхнего левого угла.

Тип: строка

```
videoScreenSize: "1024x768"
```

- Частота кадров по умолчанию составляет `12` кадров в секунду. Capability `videoFrameRate` позволяет изменить это значение:

  Тип: целое число

  ```
  videoFrameRate: 24
  ```

- По умолчанию для вывода видео Moon использует кодек `libx264`. Если этот кодек использует слишком много процессорного времени вы можете изменить его с помощью capability `videoCodec`:

  Тип: строка

  ```
  videoCodec: "mpeg4"
  ```

- Для организации своей иерархии хранения в S3 для загружаемых видео используйте capability `pattern` (или `s3KeyPattern`):

  Тип: строка

  ```
  pattern: "$quota/$browserName/$sessionId"
  ```

Сбор сетевой активности веб-страницы: enableHAR

1. Данная функциональность доступна начиная с Moon версии 2.7.2.
2. Использование данной капабилити требует предварительной настройки S3 хранилища.

В некоторых случаях вам может потребоваться собрать все HTTP-запросы, отправляемые браузером в течение Selenium сессии. Moon может автоматически сохранять весь сетевой трафик браузероа в так называемый HAR файл и загружать его в S3 хранилище. Для того чтобы включить сбор сетевой активности браузерной сессии, добавьте следующую капабилити:

Тип: boolean

```
enableHAR: true
```

Переменные окружения в сессии: env

Иногда вам может потребоваться задать переменные окружения для каждого тестового сценария (например для тестирования разных локалей). Используйте capability :

Тип: массив, формат: <ключ>=<значение>

```
env: ["LANG=ru_RU.UTF-8", "LANGUAGE=ru:en", "LC_ALL=ru_RU.UTF-8"]
```

Переменные окружения, заданные таким образом, добавляются к переменным, заданным в конфигурации Moon. Для использования этой capability в строго типизированных языках типа Java или C# используйте List:

Задание переменных env в Java

```
capabilities.setCapability("moon:options", Map.of(
    "env", Arrays.asList("LANG=ru_RU.UTF-8", "LANGUAGE=ru:en", "LC_ALL=ru_RU.UTF-8")
));
```

Дополнительные записи в /etc/hosts: hosts

Для каждого образа вы можете сконфигурировать отдельный список псевдонимов имени хоста в файле `/etc/hosts` в browsers set. Но иногда вам необходимо добавить еще несколько значений для конкретного тестового сценария. Это можно сделать следующим образом:

Тип: массив, формат: <имя-хоста>:<ip-адрес>

```
hosts: ["example.com:192.168.0.1", "test.com:192.168.0.2"]
```

Значения, заданные таким образом, переопределяют значения в `/etc/hosts`.

Используемые DNS сервера: nameservers

По умолчанию браузерные поды используют DNS сервера, заданные в Kubernetes. Иногда вам может потребоваться переопределить эти значения для конкретного тестового сценария. Это можно сделать следующим образом:

Тип: массив, формат: <ip-адрес-dns-сервера>

```
nameservers: ["192.168.0.1", "192.168.0.2"]
```

Переопределение таймаута сессии: sessionTimeout

Иногда вам может потребоваться изменить максимальное время простоя (idle timeout) для некоторых браузерных сессий. Это можно сделать при помощи следующей capability:

Тип: строка

```
sessionTimeout: "1m30s"
```

Таймаут всегда указывается в формате Golang - `30s` or `2m` or `1h2m30s` и тому подобное.

Эмуляция мобильных устройств: mobileDevice

Следующая capability позволяет сконфигурировать эмуляцию мобильных устройств:

Type: object

```
"mobileDevice": {
    "deviceName": "Apple iPhone XR",
    "orientation": "landscape"
}
```

Выбрать устройство для эмуляции можно с помощью ключа `deviceName`:

Тип: строка

```
deviceName: "Apple iPhone XR"
```

Задать нужную ориентацию устройства (альбомную или портретную) можно с помощью ключа `orientation`:

Тип: строка

```
orientation: "landscape"
```

Возможные значения ключа `orientation`: `portrait`, `vertical` (то же что и `portrait`), `landscape`, `horizontal` (то же что и `landscape`). В строго типизированных языках типа Java или C# используйте Map:

Задание mobileDevice в Java

```
capabilities.setCapability("moon:options", Map.of(
    "mobileDevice", Map.of(
        "deviceName": "Apple iPhone XR",
        "orientation": "landscape"
    )
));
```

Метки подов: labels

В некоторых случаях возникает необходимость передать дополнительные метаданные к каждой браузерной сессии, например, окружение, информацию из системы контроля версий, номер сборки, название проекта и так далее. Эти метки могут быть использованы

для сбора различной статистики по использованию браузеров.

Тип: ассоциативный массив, формат: "<ключ>": "<значение>"

```
labels: {"project": "MyCoolProject", "build-number": "14353"}
```

Метки, заданные этой capability переопределяют метки из описания конфигурации браузера. Больше информации о метках доступно по ссылке Using Custom Kubernetes Labels.

Уровень логирования браузерной сессии: logLevel

Эта функция доступна начиная с версии Moon 2.2.0 и выше.

По умолчанию логирование браузерных сессий в Moon очень ограничено с целью снизить нагрузку на кластер и системы логирования в целом. Вы можете поменять уровень логирования с помощью capability `logLevel`:

Тип: строка

```
logLevel: "INFO"
```

Список поддерживаемых браузеров: Google Chrome, Microsoft Edge, Opera и Firefox. Возможные значения для этой capability зависят от браузера:

Table 5. Поддерживаемые уровни логирования для Chrome, Microsoft Edge и Opera

Значение капабилити
ALL
DEBUG
INFO
WARNING
SEVERE
OFF

Table 6. Поддерживаемые уровни логирования для Firefox

Значение капабилити
fatal
error
warn
info
config
debug
trace

Подключение дополнительных шрифтов: additionalFonts

1. Эта функциональность доступна начиная с Moon версии 2.2.1.
2. При запуске сессии для ручного тестирования из Moon UI эта функциональность включается автоматически.
3. При подключении этой функциональности шрифты копируются в браузерный контейнер для каждого нового пода, как результат - браузеры могут стартовать немного медленнее.

По умолчанию браузеры запускаемые в Moon не поддерживают китайский, японский, тайский и другие азиатские языки. Подключить шрифты, содержащие японские/китайские и т.д. символы можно с помощью еще одной capability:

Тип: boolean

```
additionalFonts: true
```

Дополнительные данные для браузера: context

Эта функциональность доступна начиная с версии Moon 2.3.0.

Эта capability позволяет загружать произвольные данные на браузерный под. Запакуйте все необходимые файлы в архив `*.tar.gz`, значение capability должно содержать полный URL до места хранения архива. Более подробную информацию вы можете найти здесь.

Тип: строка

```
context: "https://example.com/browser-data.tar.gz"
```

Отключить менеджер окон: windowManager

Эта функциональность доступна начиная с версии Moon 2.7.0.

Менеджер окон - системное приложение, которое контролирует расположение и внешний вид окон в оконном графическом пользовательском интерфейсе. По-умолчанию, все браузеры запускаются с включенным менеджером окон. Данная капабилити позволяет отключить запуск менеджера окон. Основное изменение, которое при этом произойдет - у окон пропадут заголовки и рамки. Вы не сможете вручную поменять размер или переместить окно браузера при ручном тестировании, но вы по-прежнему можете это делать при помощи кода.

Тип: boolean

```
windowManager: false # По-умолчанию true
```

Автоматически выбирать сертификат клиента для взаимной TLS аутентификации: autoSelectClientCerts

1. Эта функциональность доступна начиная с версии Moon 2.7.2.
2. Данная капабилити работает только с браузером Chrome.

Некоторые тестируемые приложения сегодня используют так называемую взаимную TLS аутентификацию. Чтобы автоматически выбирать первый доступный TLS сертификат клиента, используйте капабилити:

Тип: boolean

```
autoSelectClientCerts: true
```

3.2.2. Headless режим

По умолчанию все браузеры в Moon стартуют в экранном режиме (вы можете видеть окно браузера). Большинство браузеров в наши дни поддерживают так называемый "безэккранный" ("headless") режим, когда страницы открываются в фоновом режиме и пользователь не видит никаких окон. Обычно этот режим включается флагом `--headless` в команде запуска браузера в Selenium capabilities.

Запуск Chrome в headless режиме

```
capabilities = {
    "browserName": "chrome",
    "goog:chromeOptions": {
        "args": ["--headless"]
    }
}
```

Moon автоматически распознает когда браузер стартует в headless режиме. Браузеры, запущенные в таком режиме, не используют графические компоненты типа X-сервера или оконного менеджера, соответственно Moon также не запускает эти компоненты. Благодаря наличию поддержки headless режима нет необходимости указывать капабилити `enableVNC`, чтобы отображать окно браузера в интерфейсе Moon.

3.2.3. Запись видео

Запись видео должна быть настроена при установке Moon.

Если видеозапись включена, то для того, чтобы записать видео, вам нужно добавить еще одну capability в ваш тест:

Включение видеозаписи в Selenium капабилити

```
capabilities = {
    "moon:options": {
```

```
        "enableVideo": True
    }
}
```

Чтобы поменять название видео, размер экрана, частоту кадров и так далее нужно добавить одну из соответствующих капабилити.

3.2.4. Эмуляция мобильных устройств

1. Эта функциональность доступна начиная с версии Moon 1.8.0 и выше.
2. Эмуляция работает только с Chrome.
3. Тестирование мобильных приложений невозможно.

Автоматическое тестирование мобильных платформ весьма актуально в наши дни. Использование настоящих устройств, подключенных к серверу через USB, требует слишком много усилий по настройке и поддержке. Использование эмуляторов Android требует настоящих железных серверов либо виртуальных машин с включенной вложенной виртуализацией. Запуск симуляторов iOS требует использования настоящих устройств от Apple. Даже при правильном расчете вычислительных ресурсов такие тесты выполняются медленнее и потребляют гораздо более памяти и процессорного времени чем настольные браузеры на физических машинах.

При этом основной целью тестирования является выявление как можно большего количества багов, а не сложное и дорогостоящее развертывание инфраструктуры автоматизации тестирования. В то же время, большое количество багов мобильной версии веб-приложения можно воспроизвести послав нужный HTTP заголовок User-Agent в браузер и выставив такой же размере экрана, как и в реальном мобильном устройстве. Подобная функциональность уже доступна в Chromium-based браузерах и называется **эмуляция мобильных устройств**.

Вот пример capabilities для включения этой функциональности:

Мобильная эмуляция в Java

```
ChromeOptions options = new ChromeOptions();
options.setCapability("browserVersion", "96.0");
options.setCapability("moon:options", Map.of(
    "mobileDevice", Map.of(
        "deviceName", "Apple iPhone XR",
        "orientation", "landscape",
    )));
```

Мобильная эмуляция в Python

```
capabilities = {
    "browserName": "chrome",
    "browserVersion": "96.0",
    "moon:options": {
```

```
        "mobileDevice": {
            "deviceName": "Apple iPhone XR",
            "orientation": "landscape"
        }
    }
}
```

Moon поставляется с готовым списком поддерживаемых устройств, хранящийся в объекте конфигурации устройств. Полный список доступных устройств содержится в разделе Supported Mobile Devices. Отредактируйте список для добавления вашего устройства.

3.2.5. Доступ к буферу обмена

1. Буфер обмена доступен только пока браузерная сессия запущена.
2. При использовании изображений поддерживается только формат PNG.
3. Если имеется установленный Lightning клиент доступ к буферу обмена работает сразу после установки.

Иногда вам может потребоваться взаимодействовать с буфером обмена, чтобы проверить работоспособность копирования/вставки текста. В Moon реализован специальный API для этой цели.

1. Запустите новую сессию, например с идентификатором `firefox-95-0-f2bcd32b-d932-4cdc-a639-687ab8e4f840`.
2. Для получения доступа к буферу обмена необходимо послать следующий HTTP запрос:

   ```
   $ curl -H 'Accept: application/json'
   https://moon.example.com/wd/hub/session/firefox-95-0-f2bcd32b-d932-4cdc-a639-
   687ab8e4f840/aerokube/clipboard

   {"value": "some-clipboard-value", "media": ""}
   ```

   Если буфер обмена содержит графическое изображение, ответ будет содержать байты закодированного в Base64 изображения:

```
{"value": "iVBORw0KGgoAAAAN....", "media": "image/png"}
```

1. Следующим запросом вы можете вставить в буфер обмена текстовое значение:

   ```
   $ curl -X POST -H 'Content-Type: application/json' --data '{"value": "some-
   clipboard-value"}' https://moon.example.com/wd/hub/session/firefox-95-0-f2bcd32b-
   d932-4cdc-a639-687ab8e4f840/aerokube/clipboard
   ```

2. Для того чтобы вставить в буфер обмена изображение - пошлите закодированные в

Base64 байты изображения:

```
$ curl -X POST -H 'Content-Type: application/json' --data '{"value":
"iVBORw0KGgoAAAAN....", "media": "image/png"}'
https://moon.example.com/wd/hub/session/firefox-95-0-f2bcd32b-d932-4cdc-a639-
687ab8e4f840/aerokube/clipboard
```

3.2.6. Загрузка файлов в браузер

Загрузка файлов в браузер - это встроенная в Selenium функциональность, поддерживаемая большинством Selenium клиентов. Далее показано несколько примеров как делается загрузка в других клиентах:

1. Загрузка в standard Java client

```
// Находим поле для загрузки файла
WebElement input = driver.findElement(By.cssSelector("input[type='file']"));

// Убеждаемся, что элемент отображается
((JavascriptExecutor) driver).executeScript("arguments[0].style.display = 'block';",
input);

// Настраиваем Selenium клиента для загрузки локальных файлов на удаленный сервер
driver.setFileDetector(new LocalFileDetector());

// Specify you local file path here (not path inside browser container!)
input.sendKeys("/path/to/file/on/machine/which/runs/tests");
```

Загрузка в Lightning Java client

```
// Находим поле для загрузки файла
WebElement fileInput = driver.elements().findFirst(By.cssSelector("input[type='file']
"));

// Загружаем файл
Path fileToUpload = Paths.get("/path/to/file/on/machine/which/runs/tests");
String fileRemotePath = driver.document().uploadFile(fileToUpload);

// Устанавливаем путь до файла как значение поля для загрузки файла
fileInput.sendKeys(fileRemotePath);
```

Загрузка в Python

```
from selenium.webdriver.remote.file_detector import LocalFileDetector

# ...

# Находим поле для загрузки файла
```

```
input = driver.find_element_by_css_selector("input[type='file']")

# Убеждаемся, что элемент отображается
driver.execute_script("arguments[0].style.display = 'block';", input)

# Загружаем файл
driver.file_detector = LocalFileDetector()
input.send_keys("/path/to/file/on/machine/which/runs/tests")
```

Загрузка в C#

```
// Создаем драйвер в Selenium
ChromeOptions options = new ChromeOptions();
IWebDriver driver = new RemoteWebDriver(new Uri("https://moon.example.com/wd/hub"),
options);

// Открываем страницу
driver.Navigate().GoToUrl("https://example.com/");

// Загружаем файл
IAllowsFileDetection allowsDetection = (IAllowsFileDetection)driver;
allowsDetection.FileDetector = new LocalFileDetector();
driver.FindElement(By.Id("uploadfile_0")).SendKeys("/tmp/file.txt");
```

Загрузка в Webdriver.io

```
var filePath = path.join('/path/to/file/on/machine/which/runs/tests');
var remoteFilePath = browser.uploadFile(filePath);
$("input[type='file']").setValue(remoteFilePath);
```

3.2.7. Дополнительные данные для браузера

1. Эта функциональность доступна начиная с версии Moon 2.3.0.
2. Поддерживаются только архивы упакованные в `*.tar.gz`. Поддержки `*.zip` архивов нет по причине более низкой производительности.

Часто при работе с браузером вам нужны дополнительные данные: расширения браузера, тестовые файлы, файлы настроек браузера (профиль) и так далее. В обычном Selenium такая информация передается во фрагментах кода. Важно знать что все эти данные передаются в теле HTTP запроса. Каждый раз, пересылая файл или изображение, Selenium сначала считывает данные в память, что резко повышает потребление памяти. Другой, более эффективный путь передачи этих данных - упаковка их в архив (например на стороне CI сервера). URL со ссылкой на такой архив посылается как Selenium capability, что позволяет каждой браузерной сессии загрузить данные, а только после этого запускать браузер. Такой архив мы называем `контекст браузера`, соответствующая capability называется просто `context`:

Тип: строка

```
context: https://example.com/browser-data.tar.gz
```

При использовании капабилити `context` Moon загрузит и распакует архив в директорию `/home/<user>`, где `<user>` это имя пользователя, созданного на этапе configuration object. Имя пользователя по умолчанию - `user`, соответственно директория по умолчанию - `/home/user/`.

Как распаковывается архив

```
browser-data.tar.gz    ===>     /home/user
|                               |
---- some-file.txt              ---- some-file.txt
---- some-directory             ---- some-directory
     |                               |
     ---- another-file.xpi           ---- another-file.xpi
     ---- one-more-file.png          ---- one-more-file.png
```

Как создать архив:

Создание архива с дополнительными данными для браузера

```
$ tar cvzf browser-data.tar.gz some-file.txt some-directory # Добавьте произвольное
количество файлов и каталогов с данными
```

Возможные сценарии использования этой функциональности:

1. **Загрузка файлов.** Можете упаковать любые файлы необходимые для теста и указать путь к файлу, либо открыть их в браузере:

   Капабилити для загрузки тестовых файлов в браузер

   ```
   {
     "browserName":"chrome",
     "moon:options":{"context":"https://example.com/browser-data.tar.gz"}
   }
   ```

   То же самое можно сделать запросом HTTP:

   HTTP запрос для загрузки тестовых файлов в браузер

   ```
   $ curl https://moon.example.com/wd/hub/session
   -d'{"capabilities":{"alwaysMatch":{"browserName":"chrome",
   "moon:options":{"context":"https://example.com/browser-data.tar.gz"}}}}'
   ```

   Распаковка файлов в коде Selenium:

Использование тестовых файлов из архива

```
// Находим поле для загрузки файла
WebElement input = driver.findElement(By.cssSelector("input[type='file']"));

// Указываем путь до файла из директории с контекстом
input.sendKeys("/home/user/some-directory/one-more-file.png");

// Такой файл можно и просто открыть в браузере
driver.get("file:///home/user/some-file.txt");
```

2. **Использование расширений браузера.** Упакуйте файл расширения (`extension.crx`) в архив (`extension.tar.gz`) и загрузите с помощью командной строки:

Как перепаковать расширение в `*.tar.gz`

```
$ unzip extension.crx -d extension # То же самое работает и для *.xpi файлов, т.к.
оба являются zip архивами
$ tar cvzf extension.tar.gz extension
```

Соответствующие capabilities выглядят примерно так:

Капабилити, чтобы активировать браузерное расширение

```
{
  "browserName":"chrome",
  "goog:chromeOptions":{
    "args":[
      "--disable-extensions-except=/home/user/extension",
      "--load-extension=/home/user/extension"
    ]
  },
  "moon:options":{"context":"https://example.com/extension.tar.gz"}
}
```

То же самое, но с помощью запроса HTTP:

HTTP запрос, чтобы активировать браузерное расширение

```
$ curl https://moon.example.com/wd/hub/session
-d'{"capabilities":{"alwaysMatch":{"browserName":"chrome",
"goog:chromeOptions":{"args":["--disable-extensions-
except=/home/user/extensions","--load-extension=/home/user/extensions"]},
"moon:options":{"context":"https://example.com/extensions.tar.gz"}}}}'
```

3. **Переопределение профиля браузера.** Упакуйте директорию с профилем браузера (`profile`) в архив (`profile.tar.gz`) затем загрузите с помощью команд в браузере. Соответствующая capabilities в Chrome выглядит примерно так:

Капабилити для использования готового профиля Chrome

```
{
  "browserName":"chrome",
  "goog:chromeOptions":{
    "args":["--user-data-dir=/home/user/profile"]
  },
  "moon:options":{"context":"https://example.com/profile.tar.gz"}
}
```

То же самое с помощью запроса HTTP:

HTTP запрос для использования готового профиля в Chrome

```
$ curl https://moon.example.com/wd/hub/session
-d'{"capabilities":{"alwaysMatch":{"browserName":"chrome",
"goog:chromeOptions":{"args":["--user-data-dir=/home/user/profile"]},
"moon:options":{"context":"https://example.com/profile.tar.gz"}}}}'
```

Для Firefox используйте другие аргументы командной строки:

Капабилити для использования готового профиля в Firefox

```
{
  "browserName":"firefox",
  "moz:firefoxOptions":{
    "args":["-profile","/home/user/profile"]
  },
  "moon:options":{"context":"https://example.com/profile.tar.gz"}
}
```

То же самое используя запрос HTTP:

HTTP запрос для использования готового профиля в Firefox

```
$ curl https://moon.example.com/wd/hub/session -H'Content-Type: application/json'
-d'{"capabilities":{"alwaysMatch":{"browserName":"firefox",
"moz:firefoxOptions":{"args":["-profile","/home/user/profile"]},
"moon:options":{"context":"https://example.com/profile.tar.gz"}}}}'
```

4. **Переопределение настроек пользователя.** Ранее мы объясняли что архив с контекстом браузерами распаковывается в поде браузера в домашнюю директорию. Это дает возможность переопределить некоторые конфигурационные файлы операционной системы и директории, (`~/.bashrc`, `~/.gtkrc-3.0` с целью отключить мерцание курсора, например, или директории `~/.ssh`, `~/.gpg` и так далее).

5. **Эмуляция видео с камеры.** Вы можете загрузить видео в браузерный под и использовать это в качестве готового видео с веб-камеры. Начните с подготовки готового видео:

*Преобразуем *.mp4 видео в *.y4m*

```
$ mkdir webcam-video
$ ffmpeg -i my-video.mp4 -vf hflip -pix_fmt yuv420p -s 1280x720 webcam-video/webcam-video.y4m
```

Результат упакуйте в архив:

Создаем архив с видео

```
$ tar cvzf webcam-video.tar.gz webcam-video
```

С помощью архива создайте сессию Chrome с такими capabilities:

Капабилити для эмуляции веб-камеры в Chrome

```
{
  "browserName":"chrome",
  "goog:chromeOptions":{
    "args":[
      "--disable-gpu",
      "--use-fake-ui-for-media-stream",
      "--use-fake-device-for-media-stream",
      "--use-file-for-fake-video-capture=/home/user/webcam-video/webcam-video.y4m"
    ]
  },
  "moon:options":{"context":"https://example.com/webcam-video.tar.gz"}
}
```

То же самое с помощью запроса HTTP:

HTTP запрос для эмуляции веб-камеры в Chrome

```
$ curl https://moon.example.com/wd/hub/session -H'Content-Type: application/json' -d'{"capabilities":{"alwaysMatch":{"browserName":"chrome", "goog:chromeOptions":{"args":["--disable-gpu", "--use-fake-ui-for-media-stream", "--use-fake-device-for-media-stream", "--use-file-for-fake-video-capture=/home/user/webcam-video/webcam-video.y4m"]},"moon:options":{"context":"https://example.com/webcam-video.tar.gz"}}}'
```

3.2.8. Доступ к загруженным из браузера файлам

1. Файлы доступны только при запущенной браузерной сессии.
2. В клиенте Lightning эта функциональность работает по умолчанию.

В процессе тестирования иногда необходимо загрузить файл из браузера. Чтобы проанализировать эти файлы, необходимо как-то достать их с браузерного контейнера. Для

этих целей Moon предоставляет API:

1. Запустите новую сессию, например с идентификатором `firefox-95-0-f2bcd32b-d932-4cdc-a639-687ab8e4f840`.
2. В коде тестов задайте сохранение файлов в директорию `/home/<user>/Downloads`, где `<user>` это имя пользователя сконфигурированного на этапе configuration object. Имя пользователя по умолчанию - `user`, соответственно название директории - `/home/user/Downloads`.
3. Вывести список доступных файлов можно командой:

```
curl -H 'Accept: application/json' https://moon.example.com/wd/hub/session/firefox-95-0-f2bcd32b-d932-4cdc-a639-687ab8e4f840/aerokube/download/

{"value": ["myfile.txt", "another-file.png"]}
```

4. Любой из этих файлов доступен по следующему адресу URL:

```
curl https://moon.example.com/wd/hub/session/firefox-95-0-f2bcd32b-d932-4cdc-a639-687ab8e4f840/aerokube/download/myfile.txt

file-contents-go-here
```

5. Удаление файла:

```
curl -X DELETE https://moon.example.com/wd/hub/session/firefox-95-0-f2bcd32b-d932-4cdc-a639-687ab8e4f840/aerokube/download/myfile.txt
```

6. Закройте сессию.

3.2.9. Доступ к инструментам разработчика

1. Эта функциональность работает в версии Moon 2.1.0 и выше.
2. Эта функциональность требует первоначального создания Selenium сессии. Если вы хотите использовать инструменты разработчика совместно с Puppeteer - ознакомьтесь с документацией по ссылке documentation section.
3. Для корректной работы этой функциональности нужен полный URL адрес кластера. Удостоверьтесь в Ingress настроено проксирование HTTP заголовков `X-Forwarded-Host`, `X-Forwarded-Port` и `X-Forwarded-Scheme`. Если вы используете AWS ALB или используете Moon без Ingress проксирование заголовков не производится. В этом случае вам необходимо задать Moon `-callback-url` flag, например `-callback-url https://moon.example.com/`, иначе данная функциональность будет работать неправильно.

В Selenium 4 и выше реализована bidirectional functionality, которая дает доступ к продвинутой функциональности браузеров, и это работает без необходимости дополнительной конфигурации. Пример проекта демонстрирующего как это работает доступен по ссылке.

Moon 1.x и Selenoid имеют отдельный `/devtools/` API, дающий доступ к браузеру используя Chrome Developer Tools Protocol. Для обратной совместимости это поддерживается и в версии Moon 2.x. По стандартам W3C WebDriver Selenium команды расширения Selenium должны располагаться под префиксом поставщика, поэтому, имея идентификатор сессии Selenium для доступа к этому API в Moon 2, вы должны использовать URL-адрес следующим образом:

Старое Chrome Developer Tools API для совместимости с Moon 1.x

```
wss://moon.example.com/session/<session-id>/aerokube/devtools
```

3.2.10. Изменение локали браузера

В некоторых тестовых сценариях необходимо поменять предпочтительную локаль браузера. Вы можете это сделать стандартными capabilities Selenium. Как именно поменять локаль - зависит от типа браузера.

Firefox

Изменение локали в Firefox

```
FirefoxOptions options = new FirefoxOptions();
options.setCapability("browserVersion", "75.0");
options.addPreference("intl.accept_languages", "de");
WebDriver driver = new RemoteWebDriver(new URL("https://moon.example.com/wd/hub"),
options);
```

Браузеры семейства Chromium

Изменение локали в браузерах семейства Chromium

```
ChromeOptions options = new ChromeOptions();
options.setCapability("browserVersion", "81.0");
options.setCapability("moon:options", Map.of(
    "env", Arrays.asList("LANG=de_AT.UTF-8", "LANGUAGE=at:de", "LC_ALL=de_AT.UTF-8")
));
WebDriver driver = new RemoteWebDriver(new URL("https://moon.example.com/wd/hub"),
options);
```

3.2.11. Изменение часового пояса

Проверка работоспособности приложения в различных часовых поясах является очень частым сценарием в тестировании. В зависимости от приложения для изменения часового

пояса вы можете использовать один из следующих вариантов:

Вариант 1: Задавая переменную окружения TZ

В ОС Linux изменение часового пояса делается путем задания переменной окружения TZ (от `time zone`). Чтобы сделать это в Moon вам необходимо задать env capability в вашем коде:

```
ChromeOptions options = new ChromeOptions();

capabilities.setCapability("moon:options", Map.of(
    "env", Arrays.asList("TZ=America/New_York") // Переменная TZ со значениями
наподобие "America/New_York" или "Europe/London" выставляется здесь
));

WebDriver driver = new RemoteWebDriver(new URL("https://moon.example.com/wd/hub"),
options);

driver.get("https://dateful.com/time-zone-converter"); // Пример веб-сайта, который
учитывает выбранный часовой пояс
```

Когда вы задаете часовой пояс подобным образом - тестируемое веб приложение может обновить информацию о ней используя Javascript Time API. Проблема заключается в том, что не все веб приложения используют этот API. Так что если этот вариант не сработал - пробуйте следующий.

Вариант 2: Переопределение географических координат браузера

Некоторые веб приложения выставляют часовой п оясна основании информации о геолокации браузера, с помощью Javascript Geolocation API. Если выставить часовой пояс напрямую не получается, вы можете переопределить географические координаты послав новые координаты через API:

```
ChromeOptions options = new ChromeOptions();
WebDriver driver = new RemoteWebDriver(new URL("https://moon.example.com/wd/hub"),
options);
driver = new Augmenter().augment(driver);

DevTools devTools = ((HasDevTools) driver).getDevTools();
devTools.createSession();

// Для примера используется местоположение Лондона (измените на 40.715502419712244,
-74.00597334074466 для Нью-Йорка)
devTools.send(Emulation.setGeolocationOverride(Optional.of(51.495930861102245),
        Optional.of(0.010205721644136127),
        Optional.of(1)));

driver.get("https://google.com");
WebElement element = driver.findElement(By.name("q"));
Actions actionProvider = new Actions(driver);
```

```
Action select = actionProvider
        .sendKeys("what is my time zone\n")
        .build();
select.perform();
```

Редко, но бывает что не работает ни первый ни второй вариант, это может означать что ваше веб-приложение определяет часовой пояс сопоставлением вашего IP адреса с IP адресом в базе данных провайдеров. В этом случае вам нужно сконфигурировать прокси сервер, физически расположенный в нужном регионе и настраивать браузер, чтобы он ходил через этот прокси.

3.2.12. Использование внешних хостов

Moon предполагает что большинство браузеров будут запускаться внутри подов Kubernetes или Openshift кластера. Однако, иногда необходимо запустить тесты Selenium на каких-то внешних хостах - аппаратных серверах или виртуальных машинах. В основном это необходимо по двум причинам:

1. Необходимо запустить Selenium тесты на MacOS или iOS. Согласно лицензионному соглашению MacOS и iOS должны работать на аппаратном обеспечении Apple, а запустить Kubernetes на нем довольно сложно.
2. Необходимо использовать Selenium на онлайн платформах для некоторых браузерах. В этом случае вы можете запустить часть браузеров (например Firefox, Chrome, Opera) в Moon, а те браузеры которые не работают на стандартных виртуальных машинах (например Chrome Mobile) можно запустить на реальных устройствах на внешних Selenium платформах.

Для использования внешних хостов вам необходимо следующее:

1. Набор хостов с Selenium-совместимыми решениями (Selenoid, Appium, Selenium Grid): `host1.example.com:4444`, `host1.example.com:4444` и т.д..
2. **Необязательно** VNC сервер запущенный на каждом хосте, использующий стандартный порт `5900`. На каждом VNC сервере должна быть настроена авторизация с паролем в 8+ символов.

Для каждого типа браузеров необходимо добавить следующую конфигурацию в описание конфигурации браузеров:

Шлём запрос на внешний хост

```
selenium:
    "internet explorer":
      default: 1.0.0
      repository: aerokube/moon-external-host
      env:
        - name: URLS
          value: "[\\\"http://host1.example.com:4444/\\\",
\\\"http://host2.example.com:4444/\\\"]" # Список внешних хостов
        - name: VNC_PASSWORD
```

```
        value: "myvncpassword" # Минимум 8 символов
    ]
```

При такой конфигурации запросы Selenium сессии будут случайным образом перераспределены на хосты, указанные в переменной окружения `URLS`. VNC сервер также будет работать - в веб интерфейсе Moon вы увидите экраны внешних устройств.

3.2.13. Использование прокси-серверов

Использование прокси сервера с аутентификацией при помощи имени пользователя/пароля доступно начиная с версии Moon 2.5.0.

В некоторых случаях вам может понадобиться запустить браузер через некий прокси сервер. Протокол Selenium WebDriver поддерживает стандартные capabilities для работы с прокси сервером для любого браузера. Например:

Конфигурация прокси сервера для браузера с raw capabilities

```
ChromeOptions options = new ChromeOptions();
String proxyHost = "proxy.example.com:3128";
capabilities.setCapability("proxy", Map.of(
    "proxyType", "manual",
    "httpProxy", proxyHost,
    "sslProxy", proxyHost,
));
WebDriver driver = new RemoteWebDriver(new URL("https://moon.example.com/wd/hub"),
options);
```

В некоторых языках программирования существует объект-оболочка под названием `Proxy`, который позволяет задать те же значения, но с проверкой типов:

Конфигурация прокси для браузера с объектом Proxy

```
ChromeOptions options = new ChromeOptions();
Proxy proxy = new Proxy();
String proxyHost = "proxy.example.com:3128";
proxy
    .setProxyType(Proxy.ProxyType.MANUAL)
    .setHttpProxy(proxyHost)
    .setSslProxy(proxyHost);
options.setProxy(proxy);
WebDriver driver = new RemoteWebDriver(new URL("https://moon.example.com/wd/hub"),
options);
```

Прокси сервера очень часто требуют имя пользователя и пароль. В то время как большинство реализаций Selenium не работают с такими прокси серверами, Moon позволяет сконфигурировать авторизацию сразу после установки, используя capabilities - просто добавьте имя пользователя и пароль в виде (`username:password@host:port`):

Конфигурация прокси с авторизацией

```
ChromeOptions options = new ChromeOptions();
// Note username:password on the line below
String proxyHost = "username:password@proxy.example.com:3128";
capabilities.setCapability("proxy", Map.of(
    "proxyType", "manual",
    "httpProxy", proxyHost,
    "sslProxy", proxyHost,
));
WebDriver driver = new RemoteWebDriver(new URL("https://moon.example.com/wd/hub"),
options);
```

3.3. Работа с Cypress

1. Эта функциональность доступна начиная с версии Moon 1.9.0 и выше.
2. Никаких дополнительных изменений в проекте Cypress не требуется.
3. Пример проекта с демонстрацией функциональности доступен по ссылке here.

Cypress тесты запускаются в Moon сразу после установки. Необходимые настройки:

1. Установите инструмент позволяющий запускать тесты Cypress удаленно:

```
$ npm install @aerokube/cypress-moon
```

2. Запустите ваши тесты в кластере Moon:

```
$ cd /path/to/my-test-project
my-test-project$ cypress-moon https://moon.example.com/cypress/chrome
```

Каждый вызов команды `cypress-moon` запустит новую сессию браузера в Moon.

1. Если ваша установка Moon сконфигурирована с использованием Ingress - корректный URL должен выглядеть примерно так: `https://moon.example.com/cypress/chrome`.
2. Для запуска тестов Cypress требуется отправить сжатый проект в Moon. Если проект большой и Moon работает за Ingress вам возможно потребуется увеличить максимальный размер HTTP пакета. Например, проект Nginx Ingress Controller требует следующей аннотации:

```
nginx.ingress.kubernetes.io/proxy-body-size: 128m
```

В Cypress, в отличие от Selenium нет концепции capabilities. Единственный способ запустить конкретный тип браузера или передать дополнительные значения заключается в добавлении этих значений в конечный URL. Далее описываются поддерживаемые параметры значений.

3.3.1. Выбор браузера

Вы можете запустить один из поддерживаемых в Cypress браузеров указав его название (`chrome`, `chromium`, `edge`, `electron` или `firefox`) в URL. По умолчанию Moon использует самый новый публично доступный образ браузера `browsers/cypress-<browser-name>:latest`.

Запуск Chrome (образ `quay.io/browsers/cypress-chrome:latest`)

```
$ cypress-moon https://moon.example.com/cypress/chrome
```

Запуск Chromium (образ `quay.io/browsers/cypress-chromium:latest`)

```
$ cypress-moon https://moon.example.com/cypress/chromium
```

Запуск Electron (образ `quay.io/browsers/cypress-electron:latest`)

```
$ cypress-moon https://moon.example.com/cypress/electron
```

Запуск Microsoft Edge (образ `quay.io/browsers/cypress-edge:latest`)

```
$ cypress-moon https://moon.example.com/cypress/edge
```

Запуск Firefox (образ `quay.io/browsers/cypress-firefox:latest`)

```
$ cypress-moon https://moon.example.com/cypress/firefox
```

3.3.2. Выбор определенной версии Cypress

Cypress API может меняться от версии к версии. Поэтому рекомендуется удостовериться что версия Cypress API, используемая в вашем проекте совпадает с версией использующейся в образе браузера. Чтобы использовать образ совместимый с конкретной версией Cypress - добавьте нужную версию в URL следующим образом:

Выбор образа совместимого с Cypress 7.3.0 (образ `quay.io/browsers/cypress-electron:cypress-7.3.0`)

```
$ cypress-moon https://moon.example.com/cypress/electron/cypress-7.3.0
```

3.3.3. Запись видео

Включите запись видео - просто добавьте параметр `enableVideo` в URL:

Включение записи видео

```
$ cypress-moon https://moon.example.com/cypress/electron/cypress-
7.3.0?headless=false&enableVideo=true
```

Используйте другие параметры, если вам нужно изменить название записываемого видео, разрешение экрана, частоту кадров и так далее.

3.3.4. Дополнительная функциональность

Наряду с возможностью изменить версию Cypress вы также можете включить дополнительную функциональность, например изменить разрешение экрана, задать другое название теста и так далее. Вся эта функциональность добавляется путем внесения изменений в URL:

Добавление параметров

```
$ cypress-moon https://moon.example.com/cypress/electron/cypress-
7.3.0?noExit=true&headless=false&env=LANG%3Dde_AT.UTF-8&env=LANGUAGE%3Dat:de
```

Полный список возможных параметров и описание их функциональности описан в таблице:

Table 7. Список параметров

Название параметра	Возможные значения	Значение по умолчанию	Описание
additionalFonts	`true` или `false`	`false`	Включение дополнительных шрифтов для китайского, японского, тайского и т.д. языков.
configFile	Custom Cypress configuration file	Не выставлено	Путь до конфигурационного файла Cypress configuration file. **Поддерживается версией Cypress 9.0.0 и выше.**
enableVideo	`true` or `false`	`false`	Включение записи видео.

Название параметра	Возможные значения	Значение по умолчанию	Описание
env	Переменные окружения	Не выставлено	Одна или более переменных окружения доступных для браузера. Может передаваться несколько раз: `env=LANG%3Dde_AT.UTF-8&env=LANGUAGE%3Dat:de`.
headless	`true` или `false`	`true`	Запуск браузера в headless режиме.
host	Стандартное значение из `/etc/hosts` в формате `www.example.com:127.0.0.1`	Не выставлено	Позволяет задавать дополнительные значения в файле `/etc/hosts`. Можно передавать несколько значений.
label	Метки подов Kubernetes	Не выставлено	Один или несколько меток Kubernetes которые добавляются на браузерный под. Можно задать несколько значений: `label=first-label%3Dsome-value&env=another-label%3Danother-value`.
name	Любая строка	Не выставлено	Позволяет задать имя теста (то же самое что Selenium capability под названием `name`).
nameserver	Имя DNS сервера, например `ns1.example.com`	Не выставлено	Позволяет добавлять один или несколько DNS серверов. Можно передать несколько значений.
noExit	`true` или `false`	`false`	Оставляет контейнер запущенным после завершения работы всех тестов. В основном используется для отладки.

Название параметра	Возможные значения	Значение по умолчанию	Описание
pattern	Строка с полями для заполнения	`$quota/$browserName/$sessionId`	Пользовательское значение S3 ключа pattern, используется для сохранения видео в S3.
screenResolution	`1280x1024` or `1280x1024x24`	`1920x1080x24`	Устанавливает разрешение экрана в контейнере, где запущен браузер. **Используйте существующие в Cypress способы изменения размер окна браузера.**
spec	Имя Cypress spec файла (e.g. `cypress/integration/my-spec.js`)	Не выставлено	Позволяет запускать один или несколько конкретных тестовых файлов. Можно передать несколько значений.
videoCodec	Кодек который будет использоваться для видеозаписи, например `mpeg4`	`libx264`	Позволяет поменять кодек для видеозаписи.
videoFrameRate	Четное число	12	Частота кадров в видео.
videoName	Имя файла видео с расширением	`video.mp4`	Название видео.
videoScreenSize	`1280x1024`	Значение `screenResolution`	Размер экрана записываемого видео. Если это значение меньше, чем заданное в `screenResolution`, то видео будет обрезано.

3.3.5. Запись выполненных тестов в Cypress Dashboard

Эта функциональность работает начиная с версии Cypress образов 9.6.0 и выше.

В Cypress реализована функциональность под названием Cypress Dashboard - онлайн-сервис

для хранения информации о запущенных и выполненных тестах. Чтобы послать информацию о выполненных тестах в этот сервис необходимо предоставить ключ доступа используя переменную окружения `CYPRESS_RECORD_KEY`:

Предоставление `CYPRESS_RECORD_KEY`*:*

```
$ cypress-moon https://moon.example.com/cypress/chrome/cypress-
9.6.0?&env=CYPRESS_RECORD_KEY%3Dyour-key
```

3.4. Работа с Playwright

1. Пример проекта, демонстрирующий работу с Playwright, доступен по ссылке.

Moon может запускать браузеры для Playwright сразу после установки. Пример теста Playwright который будет работать с Moon:

Пример теста Playwright для работы в Moon

```
const { firefox } = require('playwright');

(async () => {
  const browser = await firefox.connect({ timeout: 0, wsEndpoint:
'wss://moon.example.com/playwright/firefox/playwright-1.23.3' });
  const page = await browser.newPage();
  await page.goto('https://aerokube.com/moon/');
  await page.screenshot({ path: `screenshot.png` });
  await browser.close();
})();
```

Как вы можете видеть, единственное отличие от стандартного проекта Playwright — это URL-адрес веб-сокета на стороне Moon. По сравнению с Selenium в Playwright нет понятия capabilities. Единственный способ запустить браузер конкретной версии или какую-либо переменную окружения это передать параметры в URL-адрес конечной точки веб-сокета. Далее описываются конкретные параметры для передачи в URL.

Если в вашем кластере Moon использует соединение по HTTPS, а не по HTTP, соответственно и в Playwright правильный URL должен начинаться с `wss://`, а не с `ws://` (например `wss://moon.example.com/`).

3.4.1. Выбор браузера

Вы можете запустить один из поддерживаемых в Playwright браузеров указав его название (`chrome`, `chromium`, `edge`, `electron` или `firefox`) в URL. По умолчанию для загрузки образов Moon использует репозиторий `quay.io/playwright-<browser-name>`. Playwright API может меняться от версии к версии. Поэтому рекомендуется удостовериться что клиентская версия Playwright, используемая в вашем коде, совпадает с серверной версией Playwright использующейся в

образе браузера. Чтобы использовать образ совместимый с конкретной версией Playwright - добавьте нужную версию в URL следующим образом:

Запуск Chromium (образ `quay.io/browser/playwright-chromium:playwright-1.23.3`*)*

```
wss://moon.example.com/playwright/chromium/playwright-1.23.3
```

Запуск Chrome (образ `quay.io/browser/playwright-chrome:playwright-1.23.3`*)*

```
wss://moon.example.com/playwright/chrome/playwright-1.23.3
```

Запуск Firefox (образ `quay.io/browser/playwright-firefox:playwright-1.23.3`*)*

```
wss://moon.example.com/playwright/firefox/playwright-1.23.3
```

Запуск Webkit (образ `quay.io/browser/playwright-webkit:playwright-1.23.3`*)*

```
wss://moon.example.com/playwright/webkit/playwright-1.23.3
```

3.4.2. Запись видео

Включите запись видео - просто добавьте параметр `enableVideo` в URL:

Запись видео

```
wss://moon.example.com:4444/playwright/firefox/playwright-
1.23.3?headless=false&enableVideo=true
```

Используйте другие параметры, если вам нужно изменить название записываемого видео, разрешение экрана, частоту кадров и так далее.

3.4.3. Дополнительная функциональность

Как и в Selenium вы можете настроить автоматическую загрузку произвольных файлов в браузерные поды в виде архива и распаковку этого архива в нужное место. Подробную информацию можно найти здесь. Основное отличие в Playwright заключается в том что URL архива передается в параметре `context` и соответственно должен быть URL-закодирован.

Включение контекста

```
wss://moon.example.com:4444/playwright/chrome/playwright-
1.23.3?context=http%3A%2F%2Fexample.com%2Fbrowser-data.tar.gz
```

Пример подключения расширения браузера:

Подключение расширения браузера

```
var browser = await chromium.connect({ timeout: 0, wsEndpoint:
'wss://moon.example.com/playwright/chrome/playwright-
1.23.3?headless=false&context=https%3A%2F%2Fexample.com%2Fextensions.tar.gz&arg=--
disable-extensions-except%3D%2Fhome%2Fuser%2Fextensions&arg=--load-
extension%3D%2Fhome%2Fuser%2Fextensions' });
```

3.4.4. Дополнительная функциональность

Наряду с возможностью изменить версию Playwright вы также можете включить дополнительную функциональность, например изменить разрешение экрана, задать другое название теста и так далее. Вся эта функциональность добавляется путем внесения изменений в URL:

Добавление параметров

```
wss://moon.example.com/playwright/chrome/playwright-1.23.3?headless=false&arg=--use-gl
```

Полный список возможных параметров и описание их функциональности описан в таблице.

Table 8. Список параметров

Название параметра	Возможные значения	Значение по умолчанию	Описание
additionalFonts	`true` или `false`	`false`	Включение additional fonts для китайского, японского, тайского и тд языков.
arg	Аргументы командной строки браузера	Не задано	Одно или несколько значений передаваемых командной строке браузера. Значение можно передать несколько раз: `arg=--use-fake-ui-for-media-stream&arg=--use-gl`.

Название параметра	Возможные значения	Значение по умолчанию	Описание
context	HTTP URL к контексту браузера	Не задано	HTTP URL адрес до архива `*.tar.gz` с дополнительными файлами которые при необходимости можно сделать доступными для браузера (так называемый `browser context`).
devtools	`true` или `false`	`false`	Отображение Chrome Developer Toolbar панели инструментов (относится только к браузеру `chromium`).
enableVideo	`true` или `false`	`false`	Запись видео.
env	Переменные окружения	Не задано	Одна или более переменных окружения доступных для браузера. Может передаваться несколько раз: `env=LANG%3Dde_AT.UTF-8&env=LANGUAGE%3Dat:de`.
headless	`true` или `false`	`true`	Запуск браузера в headless режиме.
host	Стандартное значение из `/etc/hosts` в формате `www.example.com:127.0.0.1`	Не задано	Позволяет задавать дополнительные значения в файле `/etc/hosts`. Можно передавать несколько значений.
label	Метки подов Kubernetes	Не задано	Один или несколько меток Kubernetes которые добавляются на браузерный под. Можно задать несколько значений: `label=first-label%3Dsome-value&env=another-label%3Danother-value`.

Название параметра	Возможные значения	Значение по умолчанию	Описание
name	Любая строка	Не задано	Позволяет задать имя теста (то же самое что Selenium capability под названием `name`).
nameserver	Имя DNS сервера, например `ns1.example.com`	Не задано	Позволяет добавлять один или несколько DNS серверов. Можно передать несколько значений.
pattern	Строка с полями для заполнения	`$quota/$browserName/$sessionId`	Пользовательское значение S3 ключа pattern, используется для сохранения видео в S3.
screenResolution	`1280x1024` или `1280x1024x24`	`1920x1080x24`	Устанавливает разрешение экрана в контейнере, где запущен браузер. **Используйте существующие в Playwright способы изменения размер окна браузера.**
videoCodec	Кодек, который будет использоваться для видеозаписи, например `mpeg4`	`libx264`	Позволяет поменять кодек для видеозаписи.
videoFrameRate	Четное число	12	Частота кадров в видео.
videoName	Имя файла видео с расширением	`video.mp4`	Название видео.
videoScreenSize	`1280x1024`	Значение `screenResolution`	Размер экрана записываемого видео. Если это значение меньше чем заданное в `screenResolution` то видео будет обрезано.

3.5. Использование инструментов разработчика в Chrome

1. Эта функциональность доступна начиная с версии Moon 1.7.0.
2. Эта функциональность работает только с Chrome версии 63 и выше.
3. Мы рекомендуем всегда использовать самую последнюю версию Chrome.
4. Пример проекта доступен по ссылке.

Moon может запускать браузеры для использования инструментов разработчика по Chrome Developer Tools Protocol. Это дает возможность запускать тесты параллельно, используя библиотеки типа Puppeteer или Taiko. Для того чтобы запустить браузер с поддержкой этих инструментов, вам нужно задать URL следующим образом:

```
wss://moon.example.com/devtools/chrome
```

Если в вашем кластере Moon использует соединение по HTTPS, а не по HTTP, правильный URL должен начинаться с `wss://`, а не с `ws://` (например `wss://moon.example.com/`).

Пример теста Puppeteer показан ниже:

Доступ к API инструментов разработчика с помощью Puppeteer

```
const puppeteer = require('puppeteer-core');
const host = 'moon.example.com';
(async () => {
    const devtools = await puppeteer.connect(
        { timeout: 0, browserWSEndpoint: `wss://${host}/devtools/chrome` }
    ); // Для каждого вызова этого метода запускается новый браузер
    const page = await devtools.newPage();
    await page.goto('https://aerokube.com');
    await page.screenshot({path: 'screenshot.png'});
    const title = await page.title();

    console.log(title);

    await devtools.close();
})();
```

3.5.1. Запуск нужного браузера

Нужную версию браузера можно задать в URL:

Выбор образа Chrome версии 85 (образ `quay.io/browser/devtools-google-chrome-stable:85.0` image)

```
wss://moon.example.com/devtools/chrome/85.0
```

3.5.2. Запись видео

Включите запись видео video recording - просто добавьте параметр `enableVideo` в URL:

Запись видео

```
wss://moon.example.com/devtools/chrome/85.0?headless=false&enableVideo=true
```

Используйте другие параметры если вам нужно изменить название записываемого видео, разрешение экрана, частоту кадров и так далее.

3.5.3. Дополнительная функциональность

Дополнительную функциональность можно включить путем внесения изменений в URL:

Изменение параметров для включения дополнительной функциональности

```
wss://moon.example.com/devtools/chrome?headless=false&nameserver=ns1.example.com
```

Table 9. Список поддерживаемых параметров и описание их функциональности

Название параметра	Возможные значения	Значение по умолчанию	Описание
additionalFonts	`true` или `false`	`false`	Включение additional fonts для китайского, японского, тайского и тд языков.
arg	Аргументы командной строки браузера	Не задано	Одно или несколько значений передаваемых командной строке браузера. Значение можно передать несколько раз: `arg=--use-fake-ui-for-media-stream&arg=--use-gl`.
devtools	`true` или `false`	`false`	Отображение панели инструментоов Chrome Developer Toolbar.
enableVideo	`true` или `false`	`false`	Включение видеозаписи

Название параметра	Возможные значения	Значение по умолчанию	Описание
env	Переменные окружения	Не задано	Одна или более переменных окружения доступных для браузера. Может передаваться несколько раз: `env=LANG%3Dde_AT.UTF-8&env=LANGUAGE%3Dat:de`.
headless	`true` или `false`	`true`	Запуск браузера в headless режиме.
host	Стандартное значение из `/etc/hosts` в формате `www.example.com:127.0.0.1`	Не задано	Позволяет задавать дополнительные значения в файле `/etc/hosts`. Можно передавать несколько значений.
label	Метки подов Kubernetes	Не задано	Один или несколько меток Kubernetes которые добавляются на браузерный под. Можно задать несколько значений: `label=first-label%3Dsome-value&env=another-label%3Danother-value`.
name	Любая строка	Не задано	Позволяет задать имя теста.
nameserver	Имя DNS сервера, например `ns1.example.com`	Не задано	Позволяет добавлять один или несколько DNS серверов. Можно передать несколько значений.
pattern	Строка с полями для заполнения	`$quota/$browserName/$sessionId`	Пользовательское значение S3 ключа pattern, используется для сохранения видео в S3.
screenResolution	`1280x1024` или `1280x1024x24`	`1920x1080x24`	Устанавливает разрешение экрана в контейнере, где запущен браузер.

Название параметра	Возможные значения	Значение по умолчанию	Описание
videoCodec	Кодек, который будет использоваться для видеозаписи, например `mpeg4`	`libx264`	Позволяет поменять кодек для видеозаписи.
videoFrameRate	Четное число	12	Частота кадров в видео.
videoName	Имя файла видео с расширением	`video.mp4`	Название видео.
videoScreenSize	`1280x1024`	Значение `screenResolution`	Размер экрана записываемого видео. Если это значение меньше чем заданное в `screenResolution` то видео будет обрезано.

4. Конфигурация

4.1. Лицензионный ключ

1. Согласно лицензионному соглашению бесплатно вы можете использовать **до четырех параллельных сессий** без ограничений по времени. Если вам нужно больше параллельных сессий - закажите лицензионный ключ. Этот раздел описывает как просматривать и устанавливать лицензионный ключ. Пробный лицензионный ключ на большее количество сессий с ограниченным сроком действия можно сгенерировать на сайте Moon website.

Обычный лицензионный ключ представляет собой текстовый файл с расширением **.key**, выглядит он примерно так:

```
$ cat license.key
MG1RSVdpc2Z6YjdQQVZjd2lpei9KMkd1T3dzMTFuL1dlRjVSc3NOMUcxZk9QaUxWa3Q5SnBIakIxa09wWm0vVF
JqQ0tsa21xVG1OODVRZnlQbjBjVmRHVWFLampTOFF1a3VLRXRPcEUwbnEySG16QWFQWHRDYTVjMm9jZzZFaUJq
eFd5ODE4UFBHZzNCNWpCYXlha3oweFBscFl1RnB0V0U1Q3FwOGl5VDdKTk9abG5aSmlPdnRmZDFvSG1nNnVwVX
BLV2E4RmYwWHcreERIR29ZTE1XTldPb1hvT2ZCUnZpcDhPWW05a1FqN0hBWWVOYUtLT1lPWlVJa1dsb1gxdjNO
T1htTFpZalhsQ3h1Q3V6NWhiQjIwSjVIY0JTYnZybm9zYm14RXFkSFpQWVBKWUlKTzZvVlBnODhQeFErZ1EyTk
5sWG82TC9XeXU3aisrNU0rSEdPcXlOSEdlNGx4Zm1nNVhjMWlnNkN1OCtNSVVYRzNqUllqOUY4ZHdReWpSbFNM
NmFpL2dRQnc3TzY0U0lwdVF2d29jYi9kVzFSYWFRVkd3ZXYrOVdING8zRWRrYkVONUhRTmQ2MUxsUnFNdmtKeW
VHV21tVlVUZ2dsMDRsTFFLTmZNVG81L2JVakNBMGhNeER5VHNJdmVRRGFMMklvTWpvcFk4VERlK1U2bUJvUDVx
NVYrcCtDQVhjbjYxQlRaUVp0bmNqL0JBVkdNOEZ4NW9rWHRYSVAxUkY0a1VCckZVTDFyTWF1VkZqSk5xU1pLT2
93dUpMTTg2SEZ0Sld0eUlRK3ZZZm1pZU0xM292MnVleDBoRlhRdFkvMkt1dUhhN3dKV2pFT0pqaEVzTjhXSy82
ZlFFbi9EQzcrNkw3NzhlbmVVZ2lLZ3VFbjlMMXZMYVZ5VWtQaWc9O2V5SnNhV05sYm5ObFpTSTZJa1JsWm1GMW
```

```
JIUWlMQ0p3Y205a2RXTjBJam9pVFc5dmJpSXNJbTFoZUZObGMzTnBiMjV6SWpvMGZRPT0=
```

В первой версии Moon ключ хранился в секрете Kubernetes и монтировался в приложение как обычный файл. Во второй версии лицензионные ключи (или просто `лицензии`) хранятся в Kubernetes ресурсе.

4.1.1. Вывод списка лицензионных ключей

В отличие от других ресурсов, представленных Moon, лицензии хранятся **на уровне всего кластера**. Таким образом вам не нужно указывать имя неймспейса в следующих командах (аргумент `-n moon` указывать не нужно):

Вывести список доступных лицензий:

Список лицензий (вывод для лицензии по-умолчанию)

```
$ kubectl get licenses
NAME    LICENSEE    SESSIONS    EXPIRES    STATUS    NAMESPACE
moon    Default     4           Never      Ok        moon
```

Такой вывод вы увидите, если у вас установлена бесплатная лицензия. Значения в колонках:

- **Name**. Название лицензии.
- **Licensee**. Владелец лицензионного ключа. Обычно совпадает с названием компании, например `Acme LLC`. У бесплатной лицензии в четырьмя параллельными сессиями значение будет `Default`.
- **Sessions**. Максимальное количество браузерных сессий доступных для данной лицензии.
- **Expires**. Количество дней, через которое у лицензии закончится срок действия. Если срок действия уже закончен - значение колонки будет `Already`, а если срок действия не ограничен - значение колонки будет `Never`.
- **Status**. Состояние ключа. Принимает значения: `Ok` - активная лицензия, `Expired` - срок действия лицензии закончен, `Broken` - неверные данные лицензионного ключа.
- **Namespace**. Название неймспейса Kubernetes, где используется этот ключ.

Технически может существовать несколько ресурсов Kubernetes с названием `license`. В этом случае для работы с лицензией Moon вам нужно использовать полное название ресурса:

Вывод списка лицензий (используя полное имя ресурса)

```
$ kubectl get licenses.moon
NAME    LICENSEE    SESSIONS    EXPIRES    STATUS    NAMESPACE
moon    Default     4           Never      Ok        moon

$ kubectl get licenses.moon.aerokube.com
```

```
NAME   LICENSEE   SESSIONS   EXPIRES   STATUS   NAMESPACE
moon   Default    4          Never     Ok       moon
```

Вывод информации о лицензии в формате YAML:

Вывод информации о лицензии в формате YAML

```
$ kubectl get license moon -o yaml
apiVersion: moon.aerokube.com/v1
kind: License
metadata:
  name: moon                        ①
  # Другие метаданные Kubernetes
spec:
  data: MG1RSVdpc2Z6YjdQQV....      ②
  namespace: moon                   ③
status:
  # Другие ключи и значения
```

① Название лицензионного ключа

② Содержимое лицензионного ключа

③ Неймспейс, в котором необходимо использовать лицензионный ключ

4.1.2. Обновление лицензионного ключа

Для обновления существующего лицензионного ключа необходимо обновить информацию в поле `data` в файле ключа:

Обновление лицензионного ключа

```
$ kubectl edit license moon # В любом текстовом редакторе замените информацию в поле
data новым ключом, сохраните изменения и закройте редактор.
```

После обновления лицензионного ключа все изменения применяются автоматически. Это обычно ведет к плавному перезапуску подов Moon. При это запущенные браузерные сессии не прерываются.

4.1.3. Множество лицензионных ключей

Moon 2.x поддерживает общее использование одного лицензионного ключа на нескольких неймспейсах Kubernetes и в большинстве случаев одного ключа достаточно. Однако, в некоторых случаях вам может понадобиться отдельная установка Moon и отдельный ключ для каждой команды. Для установки двух и более ключей в разных неймспейсах вам необходимо:

1. Установить два независимых кластера Moon в неймспейсы `ns1` and `ns2`
2. Создать и сохранить файл лицензии (например `license-keys.yaml`), в которых поле `namespace` будет принимать значение `ns1` и `ns2`:

Создание ключей

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

3. Примените полученный файл:

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

- Если вы создадите два ключа с одинаковыми значениями в поле `data` - один из них будет считаться дубликатом и автоматически удалится.
- Если у вас есть два разных ключа с одинаковыми значениями в поле `namespace` - Moon будет выбирать самый последний созданный.

4. Лицензионные ключи применятся автоматически, в выводе лицензий вы увидите:

Установлено два лицензионных ключа

```
$ kubectl get licenses
NAME              LICENSEE    SESSIONS   EXPIRES   STATUS   NAMESPACE
license-key-ns1   Acme Inc.   10         32d       Ok       ns1
license-key-ns2   Acme Inc.   20         27d       Ok       ns2
```

4.1.4. Удаление лицензионного ключа

Для удаления существующего лицензионного ключа просто удалите соответствующий объект:

Удаление лицензионного ключа

```
$ kubectl delete license moon
```

После удаление последнего ключа с не пустым полем `namespace` Moon автоматически

перейдет на бесплатную схему лицензирования с четырьмя параллельными сессиями.

4.1.5. Срок действия лицензионного ключа

Существует несколько способов всегда иметь активные лицензии Moon:

Вариант 1: Проверка срока действия ключа с помощью kubectl

Самый простой способ проверки срока действия ключа/ключей - это просмотр информации о них с помощью утилиты `kubectl`:

Проверка срока действия лицензионных ключей

```
$ kubectl get licenses
NAME              LICENSEE    SESSIONS   EXPIRES   STATUS   NAMESPACE
license-key-ns1   Acme Inc.   10         32d       Ok       ns1
license-key-ns2   Acme Inc.   20         today     Ok       ns2
```

В колонке **Expires** отображается информация о днях до окончания срока действия каждой лицензии. Когда срок действия заканчивается вывод той же команды будет выыглядеть так:

Срок действия одного ключа истек

```
$ kubectl get licenses
NAME              LICENSEE    SESSIONS   EXPIRES   STATUS    NAMESPACE
license-key-ns1   Acme Inc.   10         32d       Ok        ns1
license-key-ns2   Acme Inc.   20         Already   Expired   ns2
```

Если срок действия какого либо ключа истек в колонке **Expires** вы увидите значение **Already**, статус ключа будет **Expired**.

Также, помимо `kubectl`, для отображения списка лицензионных ключей и срока их действия, вы можете использовать Kubernetes API напрямую.

Вариант 2: Используйте Prometheus метрику срока действия лицензионного ключа

Другой способ получения информации о сроке действия ключей - это встроенная в Prometheus метрика `moon_license_expire`. Подробнее об этом описано в разделе мониторинг.

Метрика срока действия лицензионного ключа в Prometheus

```
$ curl -s https://moon.example.com/metrics | grep license_expire
# HELP moon_license_expire Moon license expiration time.
# TYPE moon_license_expire gauge
moon_license_expire 1.6444512e+09
```

Данные собираются в Prometheus автоматически, вам при необходимости нужно лишь настроить графики и оповещения.

4.1.6. Обновление лицензионного ключа из внешнего секрета

В некоторых случаях вам может понадобиться хранить лицензионный ключ в секрете Kubernetes и настроить Moon брать лицензию из этого секрета. Для этой цели нами разработан компонент `license-ops`. Он представляет собой задачу Kubernetes, которая считывает содержание лицензионного ключа из секрета и автоматически обновляет ресурс Kubernetes, используемый в Moon. Включение компонента `license-ops` осуществляется установкой еще одного чарта Helm:

1. После установки Moon, создайте обычный секрет Kubernetes в неймспейсе Moon, содержащий информацию о ключе:

   Пример секрета с лицензионным ключом

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: licensekey
     namespace: moon
   stringData:
     license.key: MG1RSVdpc2Z6.... # Вставьте лицензиюнный ключ сюда
   ```

2. Установите чарт Helm:

   Установка компонента license-ops из чарта Helm

   ```
   $ helm upgrade --install -n moon license-ops aerokube/license-ops
   ```

   Изменить имя секрета, график запуска задачи и другие параметры можно с помощью параметров Helm:

   Обновление аргументов license-ops

   ```
   $ helm upgrade --install --set secretName=mysecret --set schedule="0 * * * *" -n
   moon license-ops aerokube/license-ops
   ```

4.2. Пользователи и квоты

4.2.1. Пользователи

1. По умолчанию Moon не требует аутентификации по имени пользователя и паролю. Этот раздел описывает настройку аутентификации в Moon. Обычно это требуется, если в кластере работает несколько команд и каждой необходим защищенный доступ.
2. Moon в большинстве случаев обрабатывает запросы на браузеры из кода. Этот код обычно поддерживается и выполняется командой разработчиков, таким образом использовать в коде имя пользователя и

пароль реального пользователя не является хорошим решением. Хотя имя пользователя и пароль конкретного человека могут использоваться для доступа в графический интерфейс Moon UI, программный код обычно содержит имя и пароль какого-то сервисного аккаунта или робота, который знает и использует вся команда. Соответственно, в данной документации мы будем использовать слово "имя пользователя" относительно ко всей команде так как одно имя пользователя в Moon часто используется всей командой.

3. Мы рекомендуем настроить TLS шифрование при использовании аутентификации. Иначе злоумышленник может перехватить сетевой трафик между вашим компьютером и кластером Moon и извлечь оттуда имя пользователя \ пароль.
4. Ознакомьтесь с нашей статьей о безопасности кластера. Там мы также описываем конфигурацию TLS и авторизацию.

По сравнению с первой версией Moon версии 2.x **не имеет встроенного механизма аутентификации**. Причина этого заключается в том что мы рекомендуем осуществлять аутентификацию с помощью Ingress в Kubernetes, либо иным способом. Moon берет имя пользователя из HTTP заголовка `X-Moon-Quota`, который устанавливается в Ingress, либо иным способом. Следующий раздел описывает возможные конфигурации для аутентификации.

Basic HTTP аутентификация

В настоящее время самым популярным методом авторизации в автоматизации браузеров является так называемая basic HTTP аутентификация. При использовании этого метода имя пользователя и пароль передаются в HTTP заголовке `Authorization`:

```
Authorization: Basic base64("username:password")
```

Также поддерживается передача данных аутентификации в URL вида:

```
https://username:password@example.com/
```

Настройка Ingress является самым простым способом настроить basic HTTP аутентификацию. Ниже описаны несколько возможных вариантов конфигурации в Moon.

Вариант 1. Установка Ingress Nginx из Helm чарта

1. Этот способ работает в Kubernetes, но не работает в Openshift. Инструкции для Openshift смотрите в разделе Openshift.
2. В данном примере мы настраиваем Moon для запуска браузеров в отдельном неймспейсе Kubernetes для каждого пользователя. Уже существующие варианты работы Moon описаны здесь.
3. Необходимы права доступа, позволяющие создать новый неймспейс Kubernetes. Детально необходимых в Moon правах доступа читайте здесь.

Как работает Nginx Ingress

[users-nginx-ingress]

Самый простой способ защитить многопользовательский кластер Moon это подключить Nginx Ingress, который устанавливается автоматически через чарт Helm (https://github.com/aerokube/charts):

1. Добавьте репозиторий Aerokube charts:

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

2. Создайте файл `values.yaml` с указанием хоста Ingress и списка пользователей, которых нужно создать:

```yaml
ingress:
  host: moon.example.com # Укажите имя хоста кластера
quota:
  moon: null # Отключите режим одного неймспейса в Moon
  alpha-team:
    namespace: alpha # Пароль для этой команды будем сгенерирован автоматически
  beta-team:
    namespace: beta
    password: beta-team-password # Значение пароля можно выставить и явно
```

Поле `password` может быть задано, а может оставаться пустым и в зависимости от количества пользователей конечное поведение будет разным:

Вариант 1. Один пользователь.

- поле `password` не задано: нет авторизации.
- поле `password: ''` (пустая строка): авторизация задана, пароль генерируется.
- поле `password: какой-то-пароль`: авторизация задана, пароль соответствует заданному значению.

Вариант 2. Два или больше пользователей.

- поле `password` не задано или значение пустое: авторизация сконфигурирована, пароль генерируется.
- поле `password: какой-то-пароль`: авторизация задана, пароль соответствует заданному значению.

3. Установите Moon применив ваш файл `values.yaml`:

```
$ helm upgrade --install -f values.yaml -n moon moon aerokube/moon2
```

4. Будет создан отдельный неймспейс для каждой команды:

```
$ kubectl get namespaces
NAME              STATUS   AGE
alpha              Active   40m
beta              Active   40m
# Другие неймспейсы
```

В каждом неймспейсе автоматически будет создан секрет с паролем пользователя:

```
$ kubectl get secrets -n alpha
NAME                            TYPE                                  DATA   AGE
alpha-team-basic-auth-password   Opaque                                 1      2m11s
# Другие секреты
```

Пароль хранится в объекте Secret. Для того чтобы узнать пароль, выведите значение в поле `password` (оно будет зашифровано в Base64)) и дешифруйте его. Пример команды:

```
$ kubectl get secret alpha-team-basic-auth-password -n alpha -o 'go-
template={{index .data "password"}}' | base64 -d
8X4juoCQ9gHAACqbc05B3oPXUcV6Oxb7KNTSYdM15eYF
```

5. Используйте в коде название квоты (`alpha-team`, `beta-team` и так далее) из секрета как имя пользователя и пароль с предыдущего шага. Те же учетные данные должны использоваться для доступа к графическому интерфейсу. В режиме множества неймспейсов в интерфейсе показываются только сессии конкретного пользователя.
6. Для настройки TLS шифрования используйте стандартный приватный ключ (`server.key`) и сертификат (`server.crt`) следующим образом:

```
$ helm upgrade --install -f values.yaml --set-file ingress.tlsCert=server.crt --set
-file ingress.tlsKey=server.key -n moon moon aerokube/moon2
```

Вариант 2. Ручная конфигурация Nginx Ingress

При этом варианте мы настраиваем только аутентификацию. Режим множества неймспейсов не включен.

Шаги для настройки Nginx Ingress вручную:

1. Создайте текстовый файл в формате htpasswd с указанием списка пользователей:

```
$ htpasswd -Bbn new-user new-user-password >> users.htpasswd # Как добавить нового
пользователя
$ htpasswd -Bb users.htpasswd some-user new-password # Как обновить пароль
$ htpasswd -D users.htpasswd test-user # Как удалить пользователя
```

Содержимое файла в итоге будет выглядеть примерно так:

```
$ cat users.htpasswd
alpha-team:$apr1$.dZyHlKN$jdoZkin/kPviFNArx/cVL1 # Имя пользователя alpha-team,
пароль зашифрован
beta-team:$apr1$gyqzbSpt$RBNcxrsQaolPZCQZW0VQW1
```

2. Сохраните содержимое файла в секрет Kubernetes:

```
$ kubectl create secret generic moon-basic-auth --from-file=users.htpasswd -n moon
```

3. Настройте Nginx Ingress, чтобы он использовал учетные данные для basic HTTP аутентификации:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: moon
  namespace: moon
  annotations:
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
    nginx.ingress.kubernetes.io/auth-type: basic
①
    nginx.ingress.kubernetes.io/auth-secret: moon-basic-auth
②
    nginx.ingress.kubernetes.io/auth-realm: 'Authentication Required - Moon Realm'
③
    nginx.ingress.kubernetes.io/configuration-snippet: |
④
      proxy_set_header X-Moon-Quota $remote_user;
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "108000"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "108000"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "108000"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - moon.example.com
    secretName: moon-tls
⑤
  rules:
  - host: moon.example.com
```

```
    http:
      paths:
      - path: /wd/hub
        pathType: Prefix
        backend:
          service:
            name: moon
            port:
              number: 4444
      # Other rules
```

① Тут включается basic HTTP аутентификация в Nginx

② Здесь мы настраиваем Nginx использовать наш список пользователей

③ Желаемое имя для realm

④ Тут Nginx задает заголовки `X-Moon-Quota` и `Authorization`

⑤ Конфигурация TLS описана здесь

Вариант 3. Openshift Ingress

Openshift Ingress использует HAProxy, в котором использование basic HTTP аутентификации ограничено или недоступно. Для преодоления этого ограничения Moon предоставляет дополнительный контейнер под названием moon-basic-auth. Этот контейнер читает список пользователей из файла htpasswd, проверяет учетные данные пользователя из заголовка `Authorization` и посылает в Moon корректный заголовок `X-Moon-Quota`.

Как работает аутентификация в Openshift

[users-openshift]

Настройка пользователей в Openshift делается через чарт Helm точно таким же способом, как и в Nginx Ingress. Вам нужно лишь добавить параметр `openshift: true` в файл `values.yaml` следующим образом:

1. Создайте файл `values.yaml`:

```
ingress:
  host: moon.example.com
  openshift: true # Эта настройка включает дополнительный контейнер с поддержкой
аутентификации для Openshift
quota:
  moon: null
  alpha-team:
    namespace: alpha
  beta-team:
    namespace: beta
    password: beta-team-password
```

2. Установите Moon из Helm чарта:

```
$ helm upgrade --install -f values.yaml -n moon moon aerokube/moon2
$ helm upgrade --install -f values.yaml --set-file ingress.tlsCert=server.crt --set
-file ingress.tlsKey=server.key -n moon moon aerokube/moon2 # Та же самая команда с
включенным TLS шифрованием
```

Option 4. Пользовательский Ingress

Наш Helm чарт позволяет настроить Ingress произвольным образом при помощи параметра `customIngress`. При использовании пользовательского Ingress наш чарт всегда запускает отдельный контейнер под названием moon-basic-auth, так же как в случае с Openshift. Это означает что авторизация производится внутри пода Moon и любой доступ в Moon через сервисы Kubernetes будет требовать авторизацию. Вот, к примеру, конфигурация ALB Ingress в AWS:

1. Создайте файл `values.yaml`:

```yaml
customIngress:
  enabled: true
  annotations:
    external-dns.alpha.kubernetes.io/hostname: moon.example.com
    alb.ingress.kubernetes.io/group.name: moon
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
  ingressClassName: alb
  host: moon.example.com
  paths:
    - path: /api
      port: 9090
    - path: /cypress
      port: 4444
    - path: /playwright
      port: 4444
    - path: /devtools
      port: 4444
    - path: /metrics
      port: 4444
    - path: /wd/hub/session
      port: 4444
    - path: /ui
      port: 9090
    - path: /
      port: 8080
quota:
  moon: null
  alpha-team:
    namespace: alpha
  beta-team:
    namespace: beta
```

```
    password: beta-team-password
```

2. Установите Moon с помощью Helm чарта:

```
$ helm upgrade --install -f values.yaml -n moon moon aerokube/moon2
$ helm upgrade --install -f values.yaml --set-file ingress.tlsCert=server.crt --set
-file ingress.tlsKey=server.key -n moon moon aerokube/moon2 # Та же самая команда с
включенным TLS шифрованием
```

Поддержка OpenID Connect

Moon поддерживает интеграцию с решениями на базе OpenID Connect. OpenID Connect это технология на базе OAuth, предоставляющая информацию для авторизации (OAuth предоставляет только возможности авторизации). Существующие реализации OpenID Connect https://openid.net/developers/libraries/ позволяют легко делегировать авторизацию и аутентификацию сторонним провайдерам:

- Популярным провайдерам OAuth cloud: Github, Google, Microsoft, Amazon Web Services, LinkedIn, Facebook, Okta и так далее;
- Популярным корпоративным службам аутентификации: OpenLDAP, Active Directory и другим решениям на базе протокола LDAP.

Список конкретных поддерживаемых сторонних провайдеров зависит от вашей конфигурации OpenID Connect. Подробную информацию о том как взаимодействовать со сторонним провайдером обычно можно найти в настройках реализации OpenID Connect.

Moon and OpenID Connect

[moon-and-openid-connect]

Как вам может быть известно, Moon представляет собой **HTTP API** который позволяет автоматизировать браузерное тестирование в коде, а также **графический интерфейс пользователя** помогающий анализировать и отлаживать запускаемые тесты. Доступ к каждому из этих компонентов настраивается по разному:

- Доступ к **графическому интерфейсу Moon** может быть защищен реверс-проксированием, например, с помощью OAuth2 Proxy.
- Доступ к **Moon HTTP API** защищен демоном `moon-auth`, который входит в состав дистрибутива Moon. Основная причина создания отдельного демона это механизм работы браузеров - веб интерфейсы хранят учетные данные для авторизации в cookie, которые доступны только браузерам. Программы реализующие автоматизацию тестирования браузеров через Moon HTTP API не передают данные для аутентификации в cookie. Работа `moon-auth` демона заключается в преобразовании учетных данных этих программ из HTTP заголовка в формат OpenID Connect.

Защита компонентов Moon

[protecting-moon-components]

4.2.2. Квоты

Как вы возможно уже знаете, Moon является многопользовательским приложением. Для каждого пользователя необходимо создать отдельную квоту. Например, для пользователя `alice` необходимо создать квоту с таким же именем. Если имеется только одна квота - аутентификация не требуется.

Основной конфигурационный объект в Moon называется `quota`. Он содержит все настройки Moon конкретного пользователя. Вывести список доступных квот можно так:

Вывод списка квот

```
$ kubectl get quotas -n moon
NAME   NAMESPACE   CONFIG    BROWSERS   DEVICES   AGE
moon   moon        default   default    default   13h
```

Как видите, квота это встроенный объект Kubernetes, который содержит следующую информацию:

- **Namespace**. Имя неймспейса, в котором Moon запускает браузеры. По умолчанию это тот же неймспейс, где запущен Moon. Детальное описание работы с неймспейсами доступно по ссылке.
- **Config**. Название конфигурационного объекта Moon который позволяет распределять вычислительные ресурсы среди образов Moon, пользователей, групп и прочего.
- **Browsers**. Название набора браузеров, которое будет использоваться для данной квоты.
- **Devices**. Название набора устройств, которое будет использоваться для данной квоты.

Такой подход позволяет легко переиспользовать конфигурацию браузеров или конфигурацию устройств под разные квоты. Вывод списка в формате YAML:

Список квот в формате YAML

```
$ kubectl get quotas -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: Quota
  metadata:
    name: moon                    ①
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    browsers: default             ②
    config: default               ③
    devices: default              ④
    namespace: moon               ⑤
kind: List
metadata:
```

```
  # Метаданные списка
```

① Название квот

② Название набора браузеров, которое будет использоваться для данной квоты.

③ Название конфигурационного объекта которое будет использоваться для данной квоты.

④ Название набора устройств, которые будут использоваться для данной квоты.

⑤ Неймспейс, в котором Moon запускает браузеры.

Редактирование объекта квоты:

Редактирование:

```
$ kubectl edit quota.moon moon -n moon # Внесите изменения в текстовом редакторы,
сохраните и выйдите
$ kubectl edit team moon -n moon # Более короткая команда, если вы не хотите
использовать полное название объекта Kubernetes (quota.moon)
```

Здесь мы используем название `quota.moon` а не просто `quota` так как в Kubernetes название `quota` соответствует нативному ResourceQuota объекту.

4.2.3. Конфигурационный объект

Конфигурационный объект содержит различные параметры: вычислительные ресурсы, выделенные для каждого образа Moon, идентификаторы группы или пользователя для запуска подов и так далее. Этот объект соответствует файлу `service.json` в Moon версии 1.x. Для вывода списка конфигурационных объектов запустите команду:

Вывод списка конфигурационных объектов

```
$ kubectl get configs -n moon
NAME      AGE
default   2d22h
```

Вывод того же списка в формате YAML:

Вывод списка в YAML

```
$ kubectl get configs -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: Config
  metadata:
    name: default                          ①
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
```

```
  additionalTrustedCAs: |                    ②
    -----BEGIN CERTIFICATE-----
    ...
  containers:                                ③
    browser:                                 ④
      resources:                             ⑤
        limits:                              ⑥
          cpu: "1"
          memory: 2Gi
        requests:                            ⑦
          cpu: 500m
          memory: 2Gi
    ca-certs:                                ⑧
      repository: aerokube/ca-certs          ⑨
      version: 2.0.0                         ⑩
      resources:                             ⑪
        limits:                              ⑫
          cpu: 250m
          memory: 64Mi
        requests:                            ⑬
          cpu: 100m
          memory: 64Mi
      securityContext:                       ⑭
        # Параметры security context
    defender:                                ⑮
      # Такие же поля, как для ca-certs
    proxy:                                   ⑯
      # Такие же поля, как для ca-certs
    video-recorder:                          ⑰
      # Такие же поля, как для ca-certs
    vnc-server:                              ⑱
      # Такие же поля, как для ca-certs
    x-server:                                ⑲
      # Такие же поля, как для ca-certs
  group:                                     ⑳
    id: 4096
    name: user
  serviceAccountName: default
  sessionTimeout: 5m
  storage:
    accessKey: ""
    bucket: ""
    filename: ""
    endpoint: ""
    noProxy: ""
    httpProxy: ""
    httpsProxy: ""
    metadata: true
    pattern: ""
    secretKey: ""
    secretRef:
```

```
        accessKey: RootUser
        name: minio
        secretKey: RootPass
    user:
      id: 4096
      name: user
kind: List
metadata:
  # List metadata
```

① Название конфигурационного объекта

② Дополнительные корневые центры сертификации. Нужны, если используются самоподписанные TLS сертификаты. Если не настроено, то секция не отображается.

③ Конфигурация сервисных контейнеров (`ca-certs`, `defender`, `proxy`, `video-recorder`, `vnc-server`, `x-server`)

④ Конфигурация контейнера `browser`

⑤ Вычислительные ресурсы, выделенные на контейнер

⑥ Количество процессоров и памяти, выделенные на контейнер

⑦ Количество запросов к процессору и памяти, выделенные на контейнер

⑧ Конфигурация контейнера `ca-certs`

⑨ Репозиторий имиджей

⑩ Версия имиджа контейнеров. Если значение не задано - секция не отображается.

⑪ Вычислительные ресурсы, выделенные на контейнер

⑫ Лимиты процессоров и памяти, выделенные на контейнер

⑬ Requests процессоров и памяти, выделенные на контейнер (используются планировщиком Kubernetes)

⑭ Определение security context для контейнера

⑮ Конфигурация контейнера `defender`

⑯ Конфигурация контейнера `proxy`

⑰ Конфигурация контейнера `video-recorder`

⑱ Конфигурация контейнера `vnc-server`

⑲ Конфигурация контейнера `x-server`

⑳ Системная группа, использующаяся для запуска контейнеров

Идентификатор группы (`gid`)

Имя группы

Использующийся сервисный аккаунт Kubernetes

Время простоя по умолчанию

Конфигурация хранилища S3 (используется для хранения видео)

Имя системного пользователя для старта контейнеров

Идентификатор пользователя (`uid`)

Имя системного пользователя

Редактирование конфигурационного объекта:

Редактирование конфигурационного объекта

```
$ kubectl edit config default -n moon # Внесите изменения в текстовом редакторе,
сохраните и выйдите из редактора
```

4.2.4. Набор браузеров

Объект "набор браузеров" хранит информацию о списке доступных браузеров. В Moon 1.x этот объект хранится в файле `browsers.json`. Для просмотра списка доступных объектов используйте команду:

Вывод списка наборов браузеров

```
$ kubectl get browsersets -n moon
NAME      AGE
default   2d23h
```

Для вывода той же самой информации в формате YAML:

Вывод информации в формате YAML

```
$ kubectl get browsersets -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: BrowserSet
  metadata:
    name: default                                        ①
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    cypress:                                             ②
      chrome:                                            ③
        repository: quay.io/browsers/cypress-chrome      ④
      chromium:                                          ⑤
        # Такие же поля как для chrome
      edge:                                              ⑥
        # Такие же поля как для chrome
      electron:                                          ⑦
        # Такие же поля как для chrome
      firefox:                                           ⑧
        # Такие же поля как для chrome
    devtools:                                            ⑨
```

```
      chrome:
        # Такие же поля как для cypress
    playwright:                                          ⑩
      chrome:
        # Такие же поля как для cypress
      # Другие поддерживаемые типы браузеров
    selenium:                                            ⑪
      MicrosoftEdge:
        repository: quay.io/browser/microsoft-edge-beta
      chrome:
        repository: quay.io/browser/google-chrome-stable
      firefox:
        repository: quay.io/browser/firefox-mozilla-build
kind: List
metadata:
  # List metadata
```

① Имя объекта

② Информация о браузерах в Cypress

③ Информация о браузерах Cypress Chrome

④ Репозиторий для поиска образов Cypress Chrome

⑤ Информация о браузерах Cypress Chromium

⑥ Информация о браузерах Cypress Microsoft Edge

⑦ Информация о браузерах Cypress Electron

⑧ Информация о браузерах Cypress Firefox

⑨ Информация о браузерах Chrome Developer Tools

⑩ Информация о браузерах в Playwright

⑪ Информация о браузерах в Selenium

Для редактирования объекта конфигурации браузеров:

Редактирование объекта

```
$ kubectl edit browserset default -n moon # Внесите изменения в текстовом редакторе,
сохраните и выйдите из редактора
```

Если ранее вы использовали версию Moon 1.x, вы можете заметить, что объект в Moon 2.x немного отличается от файла `browsers.json` в Moon 1.x. Обычно содержимое файла выглядит так:

Файл `browsers.json` в версии Moon 1.x

```
{
  "chrome": {
    "default": "97.0",
    "versions": {
```

```
      "97.0": {
        "image": "quay.io/browsers/chrome:97.0",
        "port": "4444"
      }
    }
  }
}
```

Для каждого типа браузера и его версии вам нужно указать конкретный образ (например `browsers/chrome:97.0`) и файл содержал информацию только для браузеров в Selenium. Основная проблема в этом подходе заключается в том что каждое изменение в версии или образе требует ручного обновления в кластере Moon. Образы контейнера для разных версий одного и того же браузера обычно хранятся в одном репозитории, но с разными тегами, например:

```
quay.io/browser/google-chrome-stable:95.0 <==> Chrome 95.0
quay.io/browser/google-chrome-stable:96.0 <==> Chrome 96.0
quay.io/browser/google-chrome-stable:97.0 <==> Chrome 97.0
```

В Moon 2.x вместо копирования и правки одной и той же спецификации образа вам нужно только указать название репозитория в объекте конфигурации браузеров:

Спецификация браузеров в Moon 2.x

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
```

Этот новый формат конфигурации означает, что все образы браузера `chrome` используемые в тестах Selenium будут загружаться изз репозитория `quay.io/browser/google-chrome-stable`. Конкретный тег образа определяется при запуске браузера, например, таким образом можно задать Selenium capabilities:

```
browserName = chrome
browserVersion = 96.0
```

В данном случае Moon будет использовать образ `quay.io/browser/google-chrome-stable:96.0`. Таким же образом, редактируя URL, вы можете передать информацию в Cypress, Playwright и Chrome Developer Tools

Версии браузеров

В Moon 2.x все еще можно ограничить доступные версии браузеров, хотя эта функция не включена по умолчанию:

Ограничение списка доступных версий браузеров

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    versions: ["96.0", "97.0"]            ①
    default: "96.0"                       ②
    port: 4444                            ③
    path: "/"                             ④
```

① Список доступных версий

② Версия по умолчанию

③ Сконфигурированный в контейнере порт на который посылаются запросы (по умолчанию 4444)

④ Путь API для отправки запросов

Если задан определенный список версий Moon будет запускать только версии браузеров из этого списка. По умолчанию, если версия не задана - запустится первая доступная. Изменить версию браузера по-умолчанию можно указанием значения в поле `default`. Если поля `versions` и `default` не заданы - запустится самая последняя версия.

Вычислительные ресурсы

По умолчанию вычислительные ресурсы для каждого браузера задаются в конфигурационном объекте. Для каждого типа браузеров вы можете переопределить эти значения следующим образом:

Выделение вычислительных ресурсов для подов

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    resources:                            ①
      limits:                             ②
        cpu: "1.0"                        ③
        memory: "1Gi"                     ④
      requests:                           ⑤
        cpu: "1.0"                        ⑥
        memory: "1Gi"                     ⑦
```

① Раздел, описывающий вычислительные ресурсы

② Раздел, описывающий максимально разрешенные вычислительные ресурсы

③ Ограничение по процессорному времени

④ Ограничение по памяти

⑤ Раздел, описывающий Kubernetes requests (значения для планировщика)

⑥ Запросы по процессору

⑦ Запросы по памяти

Переменные окружения

В некоторых ситуациях вам может потребоваться задать переменные окружения для подов. Например, вы можете захотеть задать предпочтительный язык или часовой пояс для браузера при помощи переменных `LANG` или `TZ`. Чтобы задать произвольную переменную окружения, используйте обычный синтаксис Kubernetes:

Определение переменных окружения

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    env:                                  ①
      - name: TZ                          ②
        value: "Europe/Paris"
      - name: LANG
        value: "fr_FR.UTF-8"
```

① Раздел переменных окружения

② Конкретная переменная

Также поддерживается загрузка переменных окружения из полей пода, ConfigMap или Secret:

Загрузка переменных окружения

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    env:
        - name: MY_NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName # Поле из описания самого пода
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: some-secret # Имя секрета
              key: username # Название ключа
        - name: MEM_LIMIT
          valueFrom:
            resourceFieldRef:
              containerName: some-container # Имя контейнера
              resource: limits.memory # Ресурсный параметр
              divisor: 1Mi
        - name: SOME_KEY
          valueFrom:
            configMapKeyRef:
```

```
            name: some-map # Имя ConfigMap
            key: some-key # Название ключа
```

Пользовательские аннотации

Moon использует аннотации в формате YAML, так же как и сам Kubernetes.

В некоторых случаях при старте браузера вам может потребоваться добавить пользовательские аннотации Kubernetes на поды с браузерами. Чтобы добавить аннотации сразу для всех типов браузеров:

Добавление аннотаций Kubernetes сразу для всех типов браузеров

```
apiVersion: moon.aerokube.com/v1
  kind: BrowserSet
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    annotations:                        ①
      key1: "value1"                    ②
      key2: "value2"
```

① Общий раздел аннотаций

② Конкретные аннотации которые необходимо добавить

Для добавления аннотаций к какому-то одному типу браузеров:

Добавление аннотаций Kubernetes для определенного браузера

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    annotations:                        ①
      key1: "value1"                    ②
      key2: "value2"
```

① Раздел аннотаций

② Конкретные аннотации которые необходимо добавить

Некоторые аннотации Moon добавляет по умолчанию и их имя зарезервировано:

Table 10. Зарезервированные аннотации Moon

Аннотация	Значение
name	Имя сессии, определяемое в капабилити `name`

Пользовательские метки

Moon использует метки в формате YAML, так же как и сам Kubernetes.

В некоторых случаях при старте браузера вам может потребоваться добавить метку Kubernetes на поды с браузерами. Чтобы добавить метки сразу для всех типов браузеров:

Добавление меток Kubernetes сразу для всех типов браузеров

```
apiVersion: moon.aerokube.com/v1
  kind: BrowserSet
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    labels:                          ①
      key1: "value1"                 ②
      key2: "value2"
```

① Раздел меток

② Добавляемые метки

Для добавления меток к какому-то одному типу браузеров:

Добавление меток Kubernetes для определенного браузера

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    labels:                                ①
      key1: "value1"                       ②
      key2: "value2"
```

① Раздел меток

② Добавляемые метки

Некоторые метки Moon добавляет по-умолчанию и их имя зарезервировано:

Table 11. Зарезервированные в Moon названия меток

Метка	Значение
app	Уникальное имя пода
browserName	Имя браузера
browserVersion	Версия браузера
enableVNC	Включен ли VNC
moon	Системная метка, всегда выставлено в значение `browsers`

Метка	Значение
quota	Хранит информацию о пользовательских квотах
screenResolution	Хранит информацию о разрешении экрана, заданном пользователем

Node selector

Moon использует такой же node selector в формате YAML, как и Kubernetes.

Иногда вам может понадобиться запустить поды на определенных нодах Kubernetes (например, только на железных серверах). Kubernetes позволяет это сделать с помощью node selector. Для того чтобы выставить node selector для всех типов браузеров:

Добавление node selector для всех типов браузеров

```
apiVersion: moon.aerokube.com/v1
  kind: BrowserSet
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    nodeSelector:                          ①
      node-label-1: "label1-value"         ②
      node-label-2: "label2-value"
```

① Спецификация node selector

② Конкретная метка node selector, которая должна быть на node

Для конфигурации node selector для определенного типа браузера:

Определение node selector для конкретного браузера

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    nodeSelector:                          ①
      node-label-1: "label1-value"         ②
      node-label-2: "label2-value"
```

① Спецификация node selector

② Конкретная метка node selector, которая должна быть на node

Affinity

Moon использует такую же конфигурацию affinity в формате YAML как и

Kubernetes.

В дополнение к node selector вы также можете использовать все имеющиеся в Kubernetes функции affinity. Это предоставляет вам еще более продвинутые возможности настройки планировщика Kubernetes, например сопоставление нод Kubernetes со сложными логическими выражениями, предотвращение запуска некоторых помеченных подов на одной ноде с другими помеченными подами и т.д. Если вам необходимо добавить affinity для всех типов браузеров:

Настройка affinity для всех браузеров

```
apiVersion: moon.aerokube.com/v1
  kind: BrowserSet
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    affinity:                                          ①
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/e2e-az-name
              operator: In
              values:
              - e2e-az1
              - e2e-az2
```

① Настройка affinity

Для добавления affinity для конкретного типа браузера:

Добавление affinity для конкретного типа браузера

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    affinity:                                          ①
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/e2e-az-name
              operator: In
              values:
              - e2e-az1
              - e2e-az2
```

① Настройка affinity

Tolerations

Moon использует тот же механизм Taints и Tolerations в формате YAML, что и Kubernetes.

В дополнение к node selector и affinity в Kubernetes реализован механизм taints. Механизм taints позволяет адресно защитить ноду от размещения на ней подов. Если вы хотите запустить браузер на ноде с подобными ограничениями, вам необходимо добавить **tolerations**, то есть ряд условий, которые необходимо сопоставить с помеченными подами.

Чтобы настроить tolerations для всех типов браузеров:

Добавление tolerations для всех типов браузеров

```
apiVersion: moon.aerokube.com/v1
  kind: BrowserSet
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    tolerations:            ①
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoSchedule"
```

① Настройка tolerations

Для настройки tolerations для конкретного типа браузера:

Конфигурация для конкретного браузера

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    tolerations:                            ①
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoSchedule"
```

① Настройка tolerations

Сетевые настройки

Некоторые сценарии требуют дополнительных настроек сети в поде с браузером. Например, вам может потребоваться переопределить используемые DNS-сервера или значения в файле `/etc/hosts`. Это делается следующим образом:

Настройка сетевой конфигурации

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    dnsConfig:                                  ①
      nameservers:
        - 1.2.3.4
      searches:
        - ns1.svc.cluster-domain.example
        - my.dns.search.suffix
      options:
        - name: ndots
          value: "2"
        - name: edns0
    hostAliases:                                ②
      - ip: "127.0.0.1"
        hostnames:
        - "foo.local"
        - "bar.local"
      - ip: "10.1.2.3"
        hostnames:
        - "foo.remote"
        - "bar.remote"
```

① Секция настройки DNS

② Настройка `/etc/hosts`

Поля `dnsConfig` и `hostAliases` имеют тот же синтаксис, что и их аналоги в Kubernetes: (pod DNS config и host aliases).

Привилегированный режим

В некоторых случаях, например при запуске эмуляторов Android, контейнер должен быть запущен в привилегированном (`privileged`) режиме. Следующая настройка позволяет активировать этот режим:

Активация привилегированного режима

```
selenium:
  chrome:
    # Привилегированный режим может требоваться, например, для запуска Android
эмуляторов
    repository: browsers/android
    privileged: true                    ①
```

① Запуск подов в привилегированном режиме

4.2.5. Набор устройств

Moon загружает информацию о доступных мобильных устройствах для Мобильной эмуляции из объекта под названием `devices set`. В Moon 1.x этот объект хранится в файле `devices.json`. Для вывода списка доступных устройств выполните команду:

Вывод списка наборов устройств

```
$ kubectl get devicesets -n moon
NAME      AGE
default   2d23h
```

Для вывода списка в формате YAML:

Вывод наборов устройств в формате YAML

```
$ kubectl get devicesets -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: DeviceSet
  metadata:
    name: default
①
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    devices:
②
      Apple iPhone 11:
③
        height: 896
④
        pixelRatio: 2
⑤
        printVersion: true
⑥
        userAgent: user-agent-string-for-chrome-%s
⑦
        width: 414
⑧
      # Другие устройства
kind: List
metadata:
  # Метаданные списка
```

① Название объекта

② Список устройств

③ Определение конкретного устройства

④ Высота экрана устройства

⑤ Device pixel ratio (DPR)

⑥ Заменять ли версию Chrome на пользовательское значение (строка `%s` заменяется на версию Chrome)

⑦ User agent для устройства

⑧ Ширина экрана устройства

Для редактирования объекта выполните команду:

Редактирование набора устройств

```
$ kubectl edit deviceset default -n moon # В открывшемся редакторе внесите нужные
изменения, сохраните и закройте файл
```

4.3. Запись видео

1. В зависимости от конфигурации вычислительных ресурсов поддержка записи видео потребует дополнительно (к уже имеющимся) один процессор и один гигабайт памяти для каждой сессии браузера.
2. Поддержка записи видео является скорее инструментом отладки, мы не рекомендуем записывать видео на каждый тест. Запись видео существенно увеличивает расход вычислительных ресурсов, к тому же маловероятно, что кто-то будет просматривать тысячи созданных видео (особенно если тест завершился успешно).

В Moon имеется возможность записывать видео экрана браузера, в котором выполняется тест. Записанное видео может позже быть просмотрено в браузере или другой программе для воспроизведения видео. Записанный файл также можно добавить в виде вложения в отчет о прохождении теста. В браузерах, запущенных в Kubernetes или Openshift, при включенном автоматическом масштабировании сессии запускаются на случайных хостах которые появляются и исчезают в случайном порядке. Поэтому записанные видео должны сохраняться в постоянное хранилище перед тем как браузерная сессия будет закрыта. Moon позволяет автоматически загружать видео в сконфигурированное хранилище совместимое с S3. Подобные хранилища предоставляет Yandex.Cloud, AWS, Google Cloud, Microsoft Azure, Digital Ocean и множество других облачных провайдеров. Развернуть приватное хранилище совместимое с S3 вы можете с помощью Minio.

Включить возможность записи видео можно следующим образом:

1. В настройках Moon добавьте данные используемого S3 хранилища.
2. Включите запись видео при запуске теста.

4.3.1. Настройка S3 хранилища

1. Создайте S3 бакет. В нашем примере бакет называется `moon-test`. В можете создать S3-

совместимый бакет в большинстве доступных облачных платформ. Конфигурация Moon в самых распространенных платформах описана в таблице:

Table 12. Настройки S3 для самых распространенных облачных платформ:

Название платформы	Название сервиса	Точка входа	Версия
Yandex Cloud	Object Storage	`https://storage.yandexcloud.net`.	S3v4
AWS	AWS S3	Зависит от региона, как пример - может быть `https://s3.us-east-2.amazonaws.com`. Смотрите AWS документацию для получения полного списка.	S3v4
DigitalOcean	DigitalOcean Spaces	Зависит от региона, как пример - может быть `https://nyc3.digitaloceanspaces.com`. Смотрите документацию для получения полного списка.	S3v4
Google Cloud	Google Cloud Storage	`https://storage.googleapis.com`	S3v2
Microsoft Azure	Azure Blob Storage	Встроенная поддержка S3 отсутствует. Необходимо развертывать дополнительное программное обеспечение типа Minio.	S3v4

2. Доступ к S3 бакету может быть предоставлен либо с помощью статических учетных данных (access key и secret key), либо с помощью добавления ролей в облачных платформах. Этот раздел показывает как сконфигурировать статические учетные данные. Конфигурация ролей для доступа к S3 бакету доступна ниже. Как загружать статические учетные данные для доступа к S3 бакету из секрета Kubernetes описано здесь.

3. Обновите конфигурацию хранилища в конфигурационном объекте:

```
apiVersion: moon.aerokube.com/v1
```

```
  kind: Config
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    # Другие поля
    storage:
      accessKey: "AKIAXXXXXXXXXXXXXXXX" # Выставьте, если используются статические
ключи
      bucket: "moon-test"
      filename: "" #  Имя записываемого файла, например, myvideo.mp4
      endpoint: "https://s3.us-east-2.amazonaws.com"
      pattern: "" # См. ниже
      secretKey: "okUa0XXXXXXXXXXXXXXXXXXX" # Выставьте, если используются
статические ключи
    # Еще поля
```

4.3.2. Включение записи видео

Включение видеозаписи зависит от технологии автоматизации браузеров которую вы используете:

- Selenium
- Cypress
- Playwright
- Chrome Developer Tools

Пользовательская иерархия файлов в S3

По умолчанию загружаемые в S3 видео хранятся таким образом:

Настройки S3 бакета

```
\---my-bucket
    \---- <session-id>
        |---- video.mp4
```

Moon позволяет настроить пользовательскую иерархию хранения файлов в S3 с помощью **шаблонов S3** и заполнителей. Типичный шаблон ключа в S3 выглядит следующим образом:

Обычный шаблон ключа S3

```
$quota/$browserName/$browserVersion/$sessionId
```

Заполнители `$quota`, `$browserName`, `$browserVersion` и так далее будут заменены следующей информацией: имя пользователя, название браузера, версия браузера. Полученный ключ S3 будет использоваться в качестве пути, куда сохраняется видео. Список поддерживаемых

заполнителей показан в таблице:

Table 13. Заполнители ключей S3

Заполнитель	Значение
$sessionId	Заменяется ID сессии Selenium
$browserName	Заменяется значением имени браузера
$browserVersion	Заменяется значением версии браузера в Selenium
$date	Заменяется текущей датой, например `2024-02-01`
$quota	Заменяется названием квоты (например значение имени пользователя в Selenium URL)

По умолчанию шаблон ключа S3 это только `$sessionId`:

Иерархия хранения видео по умолчанию

```
my-bucket/chrome-71-0-686efb96-eabe-4435-af31-21a33c8a4c8b/video.mp4
```

Вы можете изменить шаблон ключа S3 в конфигурационном объекте следующим образом:

Изменение шаблона ключа S3

```yaml
apiVersion: moon.aerokube.com/v1
kind: Config
metadata:
  name: default
  namespace: moon
  # Другие метаданные Kubernetes
spec:
  # Другие поля
  storage:
    # Другие настройки S3
    pattern: "$quota/$browserName/$browserVersion/$sessionId"
  # Еще поля
```

Если вам необходимо определить шаблон ключа S3 для каждой сессии отдельно - используйте капабилити `pattern`. Более подробно об этом описано в разделе капабилити специфичные для Moon.

Получение учетных данных S3 из секрета Kubernetes

1. Эта функциональность доступна начиная с версии Moon 2.1.3.
2. Удостоверьтесь, что секрет Kubernetes создан в том же неймспейсе, где будут запускаться поды.

При необходимости вы также можете загрузить статические ключи для S3 из секрета Kubernetes вместо того, чтобы передавать эти конфигурационном объекте:

1. Создайте Kubernetes секрет в неймспейсе соответствующей квоты:

   Пример секрета с учетными данными S3

   ```
   $ cat secret.yaml
   ---
   apiVersion: v1
   kind: Secret
   metadata:
     name: credentials
   stringData:
     RootUser: "AKIAXXXXXXXXXXXXXXXX"
     RootPass: "okUa0XXXXXXXXXXXXXXXXXXXX"

   $ kubectl create -n moon -f secret.yaml
   secret/minio created
   ```

2. Добавьте значение поля `secretRef` field в конфигурационный объект следующим образом:

   Загрузка учетных данных S3 из секрета Kubernetes

   ```
   apiVersion: moon.aerokube.com/v1
   kind: Config
   metadata:
     name: default
     namespace: moon
     # Другие метаданные Kubernetes
   spec:
     # Другие поля
     storage:
       # Другие настройки S3
       secretRef:
         accessKey: RootUser # Имя поля секрета с access key
         name: credentials # Имя секрета из предыдущего шага
         secretKey: RootPass # Имя поля секрета с secret key
     # Другие поля
   ```

Доступ в S3 на основе ролей

Некоторые команды предпочитают раздавать доступ в S3 не с помощью статических ключей, а на основе выданных ролей. В этом разделе мы покажем как настроить доступ в S3 бакет на основе ролей в AWS.

Вариант 1. Используйте аннотации kube2iam и Kubernetes.

1. Установите kube2iam.

2. При помощи шаблона CloudFormation создайте роль IAM для доступа в S3 бакет:

```
#jinja2:trim_blocks: False
#jinja2:lstrip_blocks: False
{% set var = config.jinja_parameters %}
AWSTemplateFormatVersion: '2010-09-09'

Description: Contains infra components for Aerokube Moon
Parameters:
  TargetBucket:
    Description: Target bucket for IAM permissions
    Type: String
Resources:
  PodRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: aerokube-moon
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
        - Effect: Allow
          Action: sts:AssumeRole
          Principal:
            Service: ec2.amazonaws.com
        - Effect: Allow
          Action: sts:AssumeRole
          Principal:
            AWS: !Sub arn:aws:iam::${AWS::AccountId}:role/EKSInstanceRole
      Policies:
        - PolicyName: aerokube-moon
          PolicyDocument:
            Statement:
              - Action:
                  - s3:List*
                  - s3:Get*
                  - s3:Put*
                Effect: "Allow"
                Resource:
                  - !Sub "arn:aws:s3:::${TargetBucket}/*"
                  - !Sub "arn:aws:s3:::${TargetBucket}"
```

3. Добавьте аннотацию на неймспейс Moon:

```
annotations:
  iam.amazonaws.com/allowed-roles: |
    ["aerokube-moon"]
```

4. Добавьте аннотацию к подам с браузерами в наборе браузеров:

```yaml
apiVersion: moon.aerokube.com/v1
kind: BrowserSet
metadata:
  name: default
  namespace: moon
  # Other Kubernetes metadata
spec:
  annotations:
    iam.amazonaws.com/role: "aerokube-moon"
  # Other fields
```

Вариант 2. Добавьте роль IAM к сервисному аккаунту Moon.

1. С помощью AWS создайте роль IAM для сервисного аккаунта EKS.
2. Сконфигурируйте Moon to для использования этого аккаунта.

Загрузка видео через прокси

Эта функциональность доступна с версии Moon 2.6.0.

В некоторых средах с ограниченным доступом видео в S3 бакет необходимо загружать через прокси сервер. Чтобы сконфигурировать такой тип загрузки, обновите конфигурационый объект следующим образом:

Конфигурация прокси сервера для загрузи в S3

```yaml
apiVersion: moon.aerokube.com/v1
kind: Config
metadata:
  name: default
  namespace: moon
  # Другие метаданные Kubernetes
spec:
  # Другие поля
  storage:
    # Другие настройки S3
    noProxy: "*.example.com" # Не использовать прокси для этих хостов
    httpProxy: "proxy.example.com:3128" # Хост и порт для HTTP трафика
    httpsProxy: "proxy.example.com:3128" # Хост и порт для HTTPS трафика
  # Еще поля
```

Синтаксис этих полей соответствует переменным окружения `NO_PROXY`, `HTTP_PROXY` и `HTTPS_PROXY`.

Отключение поддержки instance metadata

Эта функциональность доступна с версии Moon 2.7.0.

По-умолчанию, Moon ожидает, что S3 клиент, используемый для загрузки видео, может обращаться к так называемому instance metadata API. Данная технология поддерживается в большинстве облачных платформ. Тем не менее в некоторых случаях данное API недоступно. В этом случае можно настроить Moon не использовать instance metadata API при загрузке видео в S3:

Отключение instance metadata для загрузок в S3

```
apiVersion: moon.aerokube.com/v1
kind: Config
metadata:
  name: default
  namespace: moon
  # Другие метаданные Kubernetes
spec:
  # Другие поля
  storage:
    # Другие настройки S3
    metadata: false # Это поле отключает использование instance metadata (по-умолчанию true)
  # Еще поля
```

4.4. Автоматическое обновление версий браузеров

Эта функциональность доступна начиная с версии Moon 2.3.0.

Moon 2.x загружает конкретный образ браузера по заранее определенному имени. Например, когда вы запрашиваете с помощью Selenium браузер `chrome 100.0`, по-умолчанию будет загружаться образ `quay.io/browser/google-chrome-stable:100.0`. Конкретный репозиторий, откуда будут загружаться образы может быть сконфигурирован в объекте набор браузеров. Однако в Moon UI список доступных версий берется из поля `versions` в наборе браузеров конкретной команды. Начиная с версии Moon 2.3.0 мы предлагаем автоматизированное решение для постоянного обновления списка версий браузера в Moon UI. Это решение называется `browser-ops` и распространяется в виде отдельного Kubernetes job, который периодически проверяет доступность версий в репозитории и обновляем версии браузеров в Moon UI.

4.4.1. Установка

Для использования этого решения вам нужно установить еще один Helm чарт.

1. Добавьте репозиторий Aerokube charts, если он не был добавлен ранее:

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

2. Установите чарт `browser-ops`:

```
$ helm upgrade --install -n moon browser-ops aerokube/browser-ops
```

4.4.2. Возможности конфигурации

Как и для всех остальных чартов Helm, параметры конфигурации чарта `browser-ops` хранятся в файле `values.yaml` и применяются следующим образом:

```
$ helm upgrade --install -f values.yaml -n moon browser-ops aerokube/browser-ops
```

- По умолчанию версии обновляются каждую ночь. Изменить расписание можно следующим образом:

  ```
  schedule: "0 */2 * * *" # Run every two hours
  ```

- По умолчанию `browser-ops` будет использовать самую последнюю доступную версию браузера. Вы можете поменять это поведение, указав все доступные версии либо какой-то определенный набор версий таким образом:

  ```
  browserImageVersions: all # Использовать все доступные версии браузеров
  ```

  ```
  browserImageVersions: 5 # Использовать 5 последних версий браузеров
  ```

- По умолчанию `browser-ops` будет использовать полное название версии, например `102.0.1245.30`, а не `102.0`. Для использования коротких имен версий укажите следующую опцию:

  ```
  browserImageTagFormat: short
  ```

  Поскольку при наличии основной версии браузера может одновременно присутствовать несколько минорных версий - при использовании коротких имен версий самая последняя из них не будет присутствовать в списке. Такое поведение позволяет не кэшировать минорные версии, которые скорее всего будут обновлены позже до стабильной версии.

- По умолчанию `browser-ops` обновляет только браузеры в наборе браузеров с именем `default`. Если у вас несколько наборов браузеров, вам будет необходимо указать каждый:

  ```
  browsersets:
  - default
  - alpha
  - beta
  ```

4.5. Использование собственного реестра контейнеров

По-умолчанию образы Moon (`aerokube/defender`, `aerokube/logger` и так далее) загружаются из публичного реестра контейнеров (container registry). Если в вашей организации в связи с ограничениями безопасности образы могут быть загружены только из внутреннего реестра (например `my-registry.example.com`), вам нужно сконфигурировать Moon для работы с таким реестром. Сделать это можно следующим образом:

1. Сконфигурируйте Kubernetes, чтобы правильно аутентифицироваться в этом реестре:

   ```
   $ 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 moon
   $ kubectl patch serviceaccount moon -p '{"imagePullSecrets": [{"name": "my-
   registry.example.com"}]}' -n moon # Используйте правильное имя service account
   ```

   Если вы работаете в Openshift, используйте следующие команды:

   ```
   $ oc 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 moon
   $ oc secrets link moon my-registry.example.com --for=pull -n moon
   ```

2. Скопируйте все нужные образы в ваш реестр:

   ```
   quay.io/browser/google-chrome-stable:96.0 => my-
   registry.example.com/browsers/chrome:96.0
   ```

3. Обновите набор браузеров для работы с вашим реестром:

   Набор браузеров для работы с частным реестром:

   ```yaml
   apiVersion: moon.aerokube.com/v1
   kind: BrowserSet
   metadata:
       name: default
       namespace: moon
       # Другие метаданные Kubernetes
   spec:
       # Здесь описаны другие инструменты
       selenium:
         chrome:
           repository: my-registry.example.com/browsers/chrome
         # Здесь описаны другие типы браузеров
   ```

4. Скопируйте нужную версию образов самого Moon в свой реестр:

```
aerokube/ca-certs:2.0.0 => my-registry.example.com/aerokube/ca-certs:2.0.0
aerokube/defender:2.0.0 => my-registry.example.com/aerokube/defender:2.0.0
aerokube/proxy:2.0.0 => my-registry.example.com/aerokube/proxy:2.0.0
aerokube/vnc-server:2.0.0 => my-registry.example.com/aerokube/vnc-server:2.0.0
aerokube/video-recorder:2.0.0 => my-registry.example.com/aerokube/video-
recorder:2.0.0
aerokube/x-server:2.0.0 => my-registry.example.com/aerokube/vnc-server:2.0.0
```

5. Переопределите служебные образы Moon в конфигурационном объекте:

```yaml
apiVersion: moon.aerokube.com/v1
kind: Config
metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
spec:
    containers:
      ca-certs:
        repository: my-registry.example.com/aerokube/ca-certs
        version: 2.0.0 # Это поле можно опустить и тогда Moon будет использовать
свою собственную версию (рекомендовано)
      defender:
        repository: my-registry.example.com/aerokube/defender
      proxy:
        repository: my-registry.example.com/aerokube/proxy
      vnc-server:
        repository: my-registry.example.com/aerokube/vnc-server
      x-server:
        repository: my-registry.example.com/aerokube/x-server
      video-recorder:
        repository: my-registry.example.com/aerokube/video-recorder
```

6. Скопируйте нужные версии основных образов Moon в ваш реестр:

```
aerokube/moon:2.0.0 => my-registry.example.com/aerokube/moon:2.0.0
aerokube/moon-conf:2.0.0 => my-registry.example.com/aerokube/moon-conf:2.0.0
aerokube/moon-ui:2.0.0 => my-registry.example.com/aerokube/moon-ui:2.0.0
```

7. В Helm чарте для запуска Moon и Moon UI используйте новые образы Moon.

4.6. Настройка таймаутов

4.6.1. Настройка таймаутов в Moon

Эти таймауты применяются только к Selenium. Playwright, Cypress и другие инструменты используются постоянное соединение, при разрыве которого сессия автоматически удаляется.

Иногда не все идет по плану: пользователь может неожиданно отсоединиться или браузерная сессия может стартовать слишком долго. Все это может привести к общей деградации кластера, поскольку браузерные поды с разорванными сессиями продолжают потреблять вычислительные ресурсы. Для предотвращения этого Moon автоматически определяет и закрывает простаивающие сессии браузера. Сессия считается простаивающей, если время между двумя HTTP запросами превышает установленный таймаут. Время простоя может быть увеличено, если тестируемые приложения загружаются слишком медленно. Таймауты простоя можно поменять в конфигурационном объекте:

Вывод значений конфигурационного объекта в формате YAML

```
$ kubectl get configs -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: Config
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    # Другие поля
    # Используйте значения наподобие 60s или 1m10s
    sessionTimeout: 5m                        ①
```

① Значение таймаута Selenium сессии

Существует несколько редко используемых значений командной строки Moon которые можно использовать для продвинутой конфигурации таймаута:

Table 14. Настройки таймаута с помощью аргументов командной строки (используется редко)

Аргумент	Значение по умолчанию	Назначение	Заметки
-delete-timeout	10 минут	Максимальное время для удаления ресурсов Kubernetes созданных для сессии.	Moon удаляет ресурсы используя Kubernetes API. По истечении этого таймаута Moon останавливает запрос и прекращает удаление ресурсов.

Аргумент	Значение по умолчанию	Назначение	Заметки
-session-attempt -timeout	30 минут	Максимальное время для старта браузерного пода.	Включает время на планирование и загрузку образа браузера. Таймаут на проксирование запроса балансировщика нагрузки должен быть больше этого значения.

4.6.2. Настройка других таймаутов

Тесты могут упасть не только из-за настроек таймаута в Moon. Типичная инсталляция Moon выглядит так:

Возможные источники таймаута

[timeouts]

В дополнение к таймаутам в Moon также существуют и другие места, где требуется правильно настроить таймаут:

1. **Таймаут на клиентской стороне.** Каждая библиотека Selenium использует HTTP-клиент с настройками таймаута запроса по умолчанию. Если вы часто видите сообщения `client disconnected` (клиент отключился до завершения обработки запроса) в логе Moon — это может быть признаком необходимости увеличения таймаутов HTTP-клиента в вашем коде.
2. **Таймаут балансировщика.** Обычно Moon запускается за балансировщиком (`LoadBalancer`, `Ingress` или `Router`), который также имеет значение таймаута запроса прокси сервера. Часто это значение равно минуте (`60 seconds`), таким образом если вы часто видите упавшие тесты с ошибкой `502 Bad Gateway` или `504 Gateway Timeout` - это также может быть признаком необходимости увеличить таймаут балансировщика. Как это сделать зависит от облачной платформы которую вы используете и типа балансировщика. Пример настройки в AWS cloud описан в разделе Connection was closed unexpectedly.
3. **Достигнута емкость кластера.** Если в логе вы часто видите сообщения `unexpected status` это может быть сигналом, что вы использовали все доступные вычислительные ресурсы (процессор и память) выделенные для неймспейса с браузерами.
4. **Фрагментация кластера.** В некоторых случаях у вас может быть достаточное количество ядер, и не все браузеры исчерпаны. Например, у вас может быть 4 доступных процессора, распределенных между 4 нодами Kubernetes (1 доступный процессор на ноду), но для запуска нового браузерного пода требуется минимум 2 процессора. В этом случае, несмотря на то что общего количества доступных процессоров достаточно для запуска пода, не существует ноды, на которой браузер мог бы запуститься. Если вы

видите много браузерных подов в статусе `Pending` - проверьте почему эти поды не стартуют, это можно сделать командой `kubectl`.

4.7. Настройка потребления ресурсов

4.7.1. Потребление ресурсов браузерами

По умолчанию в Moon определены некоторые разумные значения ресурсов, потребляемых каждым подом с браузером. Иногда вам может потребоваться переопределить эти значения. Для переопределения вычислительных ресурсов сразу для всех браузеров используйте конфигурационный объект:

```
$ kubectl get configs -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: Config
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    containers:
      browser:
        resources:        ①
          limits:
            cpu: "1"
            memory: 2Gi
      # Другие поля
```

① Настройка ресурсов для браузерных контейнеров

Для обновления значений ресурсов отредактируйте конфигурационный объект и сохраните изменения:

Редактирование конфигурационного объекта

```
$ kubectl edit config default -n moon # Обновите значения для вычислительных ресурсов,
сохраните и выйдите из редактора
```

Переопределить эти же значения для каждого типа браузера отдельно вы можете в наборе браузеров. Пример доступен тут.

4.7.2. Потребление ресурсов сервисными образами

Чтобы вывести настройки потребления ресурсов для сервисных образов, выведите содержимое конфигурационного объекта в формате YAML:

Ввод значений в формате YAML

```
$ kubectl get configs -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: Config
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    containers:
      browser:
        # Другие поля
      ca-certs:
        # Еще поля
        resources:
          limits:
            cpu: 250m
            memory: 64Mi
          requests:
            cpu: 100m
            memory: 64Mi
      defender:
        # Такие же поля как для ca-certs
      video-recorder:
        # Такие же поля как для ca-certs
      vnc-server:
        # Такие же поля как для ca-certs
      x-server:
        # Такие же поля как для ca-certs
      # И еще поля
```

Для изменения настроек вычислительных ресурсов для каждого сервисного образа отредактируйте конфигурационный объект.

4.7.3. Качество обслуживания подов

Стабильность и скорость автоматизации браузеров очень зависит от вычислительных ресурсов доступных для каждого пода. В Kubernetes реализован механизм Качество обслуживания (QoS), который определяет сколько вычислительных ресурсов выделять при старте пода. В подах Moon для стабильной работы автоматизации мы рекомендуем определять класс QoS `Guaranteed`. Вам необходимо удостовериться что значения `requests` и `limits` для процессора и памяти имеют одинаковые значения:

1. По умолчанию в Moon значения `requests` и `limits` совпадают для сервисных образов типа `defender`, `logger` и `videoRecorder`. В новых версиях Moon вы можете при желании переопределить эти значения.

2. Для браузерных контейнеров можно независимо переопределить значения `requests` и `limits`. Но в любом случае мы рекомендуем одинаковые значения для этих параметров, иначе вы рискуете столкнуться со случайным образом падающими браузерными тестами из-за нехватки вычислительных ресурсов определенных для пода.

4.8. Использование дополнительных доверенных сертификатов TLS

В корпоративных сетях в тестовом окружении часто используется TLS сертификаты, выпущенные корневыми центрами сертификации, неизвестными для браузеров. При попытке открыть HTTPS страницу используя такой сертификат ваш браузер по умолчанию будет отказывать в соединении с предупреждением "Your connection is not private" или "This connection is untrusted". В тестах Selenium для обхода этих предупреждений вы можете использовать стандартную капабилити (`acceptInsecureCerts = true`), но это не сработает если ваша страница использует механизм Strict Transport Security.

Для корректной работы с дополнительными доверенными TLS сертификатами вам необходимо добавить корневой центр сертификации в список доверенных сертификатов браузера:

1. Найдите корневой сертификат для центра сертификации (CA certificate) который используется в вашем окружении. Обычно в организации такие сертификаты выпускаются отделом информационной безопасности или системными администраторами и публично доступны в корпоративной сети. Например, ваш корневой сертификат может выглядеть так:

```
$ cat rootCA.crt
-----BEGIN CERTIFICATE-----
MIIGjzCCBHegAwIBAgIJAK1lW/5z8ZSoMA0GCSqGSIb3DQEBCwUAMIGLMQswCQYD
VQQGEwJFRTEQMA4GA1UECBMHRXN0b25pYTEQMA4GA1UEBxMHVGFsbGlubjEeMBwG
A1UEChQVQWVyb2t1YmUgU29mdHdhcmUgT8OcMRUwEwYDVQQDEwxhZXJva3ViZS5j
b20xITAfBgkqhkiG9w0BCQEWEmFkbWluQGFlcm9rdWJlLmNvbTAeFw0yMTAyMTcw
NjQ5NDJaFw0yMzEyMDgwNjQ5NDJaMIGLMQswCQYDVQQGEwJFRTEQMA4GA1UECBMH
RXN0b25pYTEQMA4GA1UEBxMHVGFsbGlubjEeMBwGA1UEChQVQWVyb2t1YmUgU29m
dHdhcmUgT8OcMRUwEwYDVQQDEwxhZXJva3ViZS5jb20xITAfBgkqhkiG9w0BCQEW
EmFkbWluQGFlcm9rdWJlLmNvbTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoC
ggIBAKdh54x9WZsSxIMfz1rFEHuJ8+3meUua0Q8cpgC/70F0G6X6BXOki0Cu7iET
6ETfirWuUdRKKGKXHLF8Fdv6WTqnLlDqgzy1Wp9DuPIgeJ+ztKZt+uJFkWjfQb9R
mn7Qs4vp/F9HTwqlTZl5jMQ+/nrcNAQeNEZ1H1AfZWAuSvrqp3rW33wl6IBZcqfD
VsMBknBKm/Zc8GpggY8NYxkfj7Jo2izwn/tV+DFgwF0pJkUrDZPPTiNW7q8Se2Vb
7tC6Iy9ZVgkH8hkrWrPzwW4zxz/d/Si7/cnn9A9+bF+pKrsHktnQ0ScDEAR5+52J
XAXkES/4pINpBcxvNUHGO6KXKH4rJVf3QvXXany0ugwVQ+QXirA6yOoY3XFgBxgU
P7Qd5pyQdVf/SwJ5Uk5Z9b2HXk8k/6jNxe1A6WiojTOnn1fD/VzOTn4xiobqNIpE
w5dUhlj/TiN+g3uGBH4BPo6IYHCmfsXFEcSZW75k7dRlZ3ZMI4k0utUVm3Y8B+TC
sj4WmwnXetFP2EMnRft7BnR13oLyzrFB8tkFafstcVoE6oR20pIBtAFxrSDWJ5dA
XdX2NGPNUCnd1RqJxu2SGA/xHHsyPT06iJeIZGUyRXmv6vBvyCkyeLtMEdq2Gzfi
MT0GtDkG5R+al/A+Ot3w3CMbMgUFrxvEhlxM1sEitclXJc4tAgMBAAGjgfMwgfAw
HQYDVR0OBBYEFBb9mCFAqV/JgmMxtwQ6UKzoLIQQMIHABgNVHSMEgbgwgbWAFBb9
```

```
mCFAqV/JgmMxtwQ6UKzoLIQQoYGRpIGOMIGLMQswCQYDVQQGEwJFRTEQMA4GA1UE
CBMHRXN0b25pYTEQMA4GA1UEBxMHVGFsbGlubjEeMBwGA1UEChQVQWVyb2t1YmUg
U29mdHdhcmUgT8OcMRUwEwYDVQQDEwxhZXJva3ViZS5jb20xITAfBgkqhkiG9w0B
CQEWEmFkbWluQGFlcm9rdWJlLmNvbYIJAK1lW/5z8ZSoMAwGA1UdEwQFMAMBAf8w
DQYJKoZIhvcNAQELBQADggIBAIUmJsxdrT8AN2yZqzI69qQKjLnDhuojdgM3XGL3
gJTldXR5OIMnw/na8WcIC3onHjgijUeEfslTIIHmNcqOd3hTfOq4Qq2/Qmpp+h1d
5dCzScrLFiDgjnzkX0VczOj/BtnZMgxx5x8YO80MMUWVEmVCk+i2bFVTypV9e4qw
1EJLmGTnKoo7l2jPHLUB5lL2LvSO4KHDhmWG5wtFg7/nd097yG5uBHda5ytbc6S8
CIS8IBJzd7TA4fr3qOhC298LMD96nJdccHqKYtlFvf9YZZ500nrA+pH6Kpo8PD67
8WiIW/CMtO0X9pxw+KRlmaDmCGGgRhvPyHoYqbX4svrca8uvErePtXIQILe/IISJ
TXLkiVsej8k3UDu77q/wX3ZdzknWakZyPj+CtYkkZL4vqkIDIFSUcXfynyDZNZEo
2d+npABzPB42+4xGZGGnFIsfuTMAgpbK8TAgPQNMIawfWTq2KhZ8MYHfPdkU3FBo
MaExr684sviAImqOotcoNQV2iMOKdwzA097jRBrfa43LhpdoWM0v7RVxB8s+kG0P
8nHOGmp6r6cIAk5hjHYAwQYiZjXuzvnFTtD9Ily63i+yVh8nRSY9NSLhpFpl4ezo
hn+savO4nm/HueAATnGR1iPlKnfXNVqQYdl+wwzqK1/3iHjzUUjyQkk0oTBk4Bez
ejbh
-----END CERTIFICATE-----
```


2. Добавьте данные сертификата в конфигурационный объект:

```yaml
apiVersion: moon.aerokube.com/v1
kind: Config
metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
spec:
  additionalTrustedCAs: |
    -----BEGIN CERTIFICATE-----
    MIIGjzCCBHegAwIBAgIJAK1lW/5z8ZSoMA0GCSqGSIb3DQEBCwUAMIGLMQswCQYD
    VQQGEwJFRTEQMA4GA1UECBMHRXN0b25pYTEQMA4GA1UEBxMHVGFsbGlubjEeMBwG
    A1UEChQVQWVyb2t1YmUgU29mdHdhcmUgT8OcMRUwEwYDVQQDEwxhZXJva3ViZS5j
    ....
```

Если нужно добавить несколько сертификатов - каждый сертификат добавляется с новой строки:

```yaml
apiVersion: moon.aerokube.com/v1
kind: Config
metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
spec:
  additionalTrustedCAs: |
    -----BEGIN CERTIFICATE-----
    MIIGjzCCBHegAwIBAgIJAK1lW/5z8ZSoMA0GCSqGSIb3DQEBCwUAMIGLMQswCQYD
    VQQGEwJFRTEQMA4GA1UECBMHRXN0b25pYTEQMA4GA1UEBxMHVGFsbGlubjEeMBwG
    A1UEChQVQWVyb2t1YmUgU29mdHdhcmUgT8OcMRUwEwYDVQQDEwxhZXJva3ViZS5j
```

```
    ...
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    MIIDBjCCAe6gAwIBAgIBATANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwptaW5p
    a3ViZUNBMB4XDTIyMDExMDEzMzgwNloXDTMyMDEwOTEzMzgwNlowFTETMBEGA1UE
    AxMKbWluaWt1YmVDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALpJ
    ...
    -----END CERTIFICATE-----
```

Добавленные сертификаты автоматически применяются к любому типу браузера и работают для загрузки видео в хранилище S3.

4.9. Расширенные настройки

В этом разделе описаны различные расширенные настройки, которые порой бывают необходимы для настройки кластера Kubernetes. Для большинства настроек, описанных ниже, Moon использует такой же YAML синтаксис, что и сам Kubernetes. Так выглядит обычный под в Kubernetes:

Обычный под Kubernetes, вывод в формате YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
  annotations:
    key1: "value1"
    key2: "value2"
  labels:
    key1: "value1"
    key2: "value2"
spec:
  containers:
  - name: app
    image: my-company/my-app:1.0.0
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Сравните вывод с одним из объектов Moon:

```yaml
apiVersion: moon.aerokube.com/v1
kind: BrowserSet
metadata:
```

```
  name: default
  namespace: moon
  # Другие метаданные Kubernetes
spec:
  annotations:
    key1: "value1"
    key2: "value2"
  labels:
    key1: "value1"
    key2: "value2"
```

4.9.1. Добавление пользовательских аннотаций Kubernetes

Эта настройка производится глобально или для конкретного типа браузера в объекте конфигурации браузеров. Подробно это описано по ссылке.

4.9.2. Добавление пользовательских меток Kubernetes

Эта настройка производится глобально или для конкретного типа браузера в объекте конфигурации браузеров. Подробно это описано по ссылке. Также вы можете переопределить метки с помощью капабилити `labels`.

4.9.3. Добавление сетевых политик

Сетевые политики — это специальные объекты Kubernetes, позволяющие контролировать правила сетевого файервола. Использовать их с Moon очень просто:

1. Создайте объект `NetworkPolicy`. Он может выглядеть так:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: moon
spec:
  podSelector:
    matchLabels: # Это правило будет применено к подам, у которых выставлена метка
role = browser
      role: browser
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: my-app
    ports:
    - protocol: TCP
      port: 6379
```

2. Добавьте пользовательские метки на браузерный под:

```
selenium:
  chrome:
    repository: quay.io/browser/google-chrome-stable
    labels:
      role: browser # Каждый под с браузером Chrome будет иметь метку role =
browser
```

4.9.4. Использование node selector

Эта настройка производится глобально или для конкретного типа браузера в объекте конфигурации браузеров. Подробно это описано по ссылке.

4.9.5. Использование аффинити

Эта настройка производится глобально или для конкретного типа браузера в объекте конфигурации браузеров. Подробно это описано по ссылке.

4.9.6. Использование Tolerations

Эта настройка производится глобально или для конкретного типа браузера в объекте конфигурации браузеров. Подробно это описано по ссылке.

4.9.7. Запуск браузеров в привилегированном режиме

Эта настройка производится глобально или для конкретного типа браузера в объекте конфигурации браузеров. Подробно это описано по ссылке.

4.9.8. Настройка пользовательского идентификатора пользователя (uid) и группы (gid) для браузерных подов

В Moon версии 2.x конфигурация для всех подов делается в конфигурационном объекте. Если вам нужно настроить идентификатор пользователя или группы для разных пользователей вам нужно создать несколько конфигурационных объектов и связать их с объектами квот. Значения по умолчанию перечислены ниже:

Table 15. Пользователи и группы в подах по умолчанию

Название	Значение
ID пользователя по-умолчанию	4096
Имя пользователя по-умолчанию	user
ID группы по-умолчанию	4096
Имя группы по-умолчанию	user

4.9.9. Настройка service account для подов

Настраивается для всех подов сразу в конфигурационном объекте.

Настройка сервисных аккаунтов для подов

```
$ kubectl get configs -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: Config
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    # Другие поля
    serviceAccountName: my-account        ①
    # Другие поля
```

① Настройка service account

4.9.10. Настройка security context для подов

Настраивается для каждого контейнера Moon в конфигурационном объекте. Синтаксис YAML syntax такой же, как и для Kubernetes.

Добавление security context в контейнеры подов

```
$ kubectl get configs -n moon -o yaml
apiVersion: v1
items:
- apiVersion: moon.aerokube.com/v1
  kind: Config
  metadata:
    name: default
    namespace: moon
    # Другие метаданные Kubernetes
  spec:
    # Другие поля
    containers:
      browser:
        # Другие поля
        securityContext:                    ①
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          privileged: false
          runAsGroup: 4096
          runAsNonRoot: true
```

```
          runAsUser: 4096
          seccompProfile:
            type: RuntimeDefault
        # Другие поля
      ca-certs:
        # Такие же поля, как для browser
      defender:
        # Такие же поля, как для ca-certs
      proxy:
        # Такие же поля, как для ca-certs
      video-recorder:
        # Такие же поля, как для ca-certs
      vnc-server:
        # Такие же поля, как для ca-certs
      x-server:
        # Такие же поля, как для ca-certs
```

① Определение контекста безопасности

4.10. Обновление версии Moon

Релизы Moon версионируются по схеме семантического версионирования (`MAJOR.MINOR.PATCH`), например 2.3.0, 2.5.3 и так далее. Основная версия (MAJOR) соответствует "поколению" Moon, в данный момент это вторая версия. Минорный компонент (MINOR) версии изменяется при добавлении/обновлении важной функциональности. Патч соответствует релизу (PATCH), который содержит в основном незначительные улучшения и исправление ошибок.

Основной способ установки Moon - используя Helm чарт:

Обычный способ установки Moon

```
$ helm upgrade --install -n moon moon aerokube/moon2
```

Обычно для обновления версии программного обеспечения с помощью Helm необходимо выполнить команду:

Обновление Moon до версии только с измененным патчем:

```
$ helm repo update # Загружает последнюю информацию о Helm чарте
$ helm upgrade --install -n moon moon aerokube/moon2 # Та же команда что и для
установки
```

Эти команды работают при изменении PATCH-версии в Moon (например с 2.5.0 на 2.5.1, 2.5.2 и так далее). Для обновления Moon на следующую MINOR версию (например с 2.4.0 на 2.5.0) процедура немного сложнее. Причина этой сложности заключается в том что Moon 2 для хранения конфигурации использует так называемые пользовательские ресурсы Kubernetes

и время от времени мы добавляем новые поля к этим ресурсам. В настоящее время Helm никогда не обновляет уже созданные custom resource definition (CRD), поэтому для обновления минорной версии вам нужно вручную выполнить следующее:

Обновление Moon с изменением custom resource definition

```
$ helm repo update
$ helm delete moon -n moon # Удаляем предыдущую версию
$ kubectl delete crd $(kubectl get crd | grep moon.aerokube.com | awk '{print $1}') #
Удаляем CRD из предыдущей версии Moon
$ helm upgrade --install -n moon moon aerokube/moon2 # Устанавливаем новую версию Moon
```

Изменения custom resource definitions обычно отражается в release notes (например тут). Если не обновить custom resource definition, Moon перестанет работать и вы увидите сообщения об ошибках типа:

```
# При обновлении в Helm
unknown field "spec.containers.proxy"

# В логах Moon:
moon: no such config: "default"
config controller: config "default": add: validate containers.proxy: value is not set:
using default
```

4.11. Мониторинг

Метрики Moon, например потребление браузеров легко визуализировать с помощью мониторинговых систем Prometheus и Grafana. Один из самых простых способов развернуть Prometheus в Kubernetes это использовать Prometheus Operator.

4.11.1. Установка

1. Moon уже должен быть запущен (например в неймспейсе `moon`).
2. Установите Prometheus и Grafana с помощью Prometheus Operator (например в неймспейс `monitoring`).

Пример команды установки с использованием Helm 3:

```
$ helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
$ helm repo update
$ helm install kube-prometheus-stack prometheus-community/kube-
prometheus-stack --create-namespace --namespace monitoring
```

4.11.2. Встроенные метрики Moon

Встроенные метрики Moon транслируются в `/metrics` HTTP API.

Получение метрик Prometheus

```
$ curl -s https://moon.example.com/metrics
# Список множества метрик будет отображен тут
```

Доступны следующие метрики:

Table 16. Встроенные в Moon метрики Prometheus

Название	Тип	Метка	Значение
moon_browser_limit	gauge	-	Максимальное количество сессий которое позволяет лицензионныый ключ
moon_browser_running	gauge	-	Общее количество запущенных броуерных сессий
moon_browser_count	gauge	quota, browserName, browserVersion	Потребление ресурсов браузерами по типу и версии браузера
moon_browser_queued	gauge	-	Общее количество браузерных запросов в очереди
moon_license_expire	gauge	-	Временная метка истечения срока действия лицензионного ключа Moon

4.11.3. Фильтрация подов по меткам

Если при установке Prometheus вы использовали параметр `kube-prometheus-stack` то вам доступен компонент kube-state-metrics. Этот компонент позволяет вам отфильтровывать поды Kubernetes по метке, аннотации, статусу, времени старта и так далее. Получить информацию о подах с помощью меток можно следующим запросом Prometheus:

```
kube_pod_labels{label_moon="browser", label_browserName="chrome",
label_browserVersion="96.0"}
```

Полный список возможных запросов находится по ссылке.

Moon может добавлять пользовательские метки на поды (например название проекта по автоматизации, тестируемую функциональность и так далее). Для всех подов это можно сделать в объекте конфигурации браузеров, Selenium capabilities и так далее. Например, после добавления метки `project="MyCoolProject"` на под вы можете настроить такую фильтрацию:

```
kube_pod_labels{label_moon="browser", label_project="MyCoolProject",
label_browserName="chrome"}
```

4.12. Логи

Несмотря на то что Moon работает сразу после установки, иногда возникает необходимость просмотреть содержимое лог файлов. Каждый компонент Moon пишет логи в стандартный поток вывода (`stdout`), то есть для просмотра лог файлов вы можете использовать известные команды `kubectl`. Все что связано с работой браузерных сессий выводится в контейнер `moon`:

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

Для просмотра логов в реальном времени при выполнении тестов добавьте флаг `-f`:

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

Для просмотра логов `moon-conf` и `moon-ui` используйте следующие команды:

```
$ kubectl logs -f -lapp=moon -c moon-conf -n moon
$ kubectl logs -f -lapp=moon -c moon-ui -n moon
```

Если браузерные поды по какой либо причине не удаляются, необходимо проанализировать содержимое лога контейнера `defender` для каждого зависшего пода:

```
$ kubectl logs chrome-73-0-ac15ffaa-e641-4c7f-a54c-f25b5be1f135 -c defender -n moon
```

В данном случае `chrome-73-0-ac15ffaa-e641-4c7f-a54c-f25b5be1f135` это ID сессии и имя пода с браузером.

4.13. Флаги командной строки (CLI)

Эти флаги передаются в файлах YAML в Kubernetes при старте кластера.

4.13.1. Флаги контейнера Moon

Поддерживаются следующие флаги:

```
-browser-limit value
    parallel browser sessions limit
-callback-url value
    moon callback url
-delete-timeout duration
    timeout to delete Kubernetes resources (default 10m0s)
-grace-period duration
    graceful shutdown period (default 5m0s)
-listen string
    host and port to listen to (default ":4444")
-moon-url value
    moon service url (default http://moon.moon:4444/wd/hub)
-session-attempt-timeout duration
    new session attempt timeout (default 30m0s)
-version
    show version and exit
```

4.13.2. Флаги контейнера Moon Auth

Поддерживаются следующие флаги:

```
-ca-cert string
    ca certificate to verify discovery cert (optional)
-client-id string
    client id (required)
-client-secret string
    client secret (required)
-discovery-url value
    oidc discovery url (required)
-fail-login-timeout duration
    request timeout (default 30s)
-grace-period duration
    graceful shutdown period (default 30s)
-group value
    allowed user groups (optional)
-ignore-case
    ignore user groups case
-listen string
    address to bind (default ":4545")
-request-timeout duration
    request timeout (default 30s)
-upstream-url value
    upstream url (default http://127.0.0.1:4444/)
-version
    show version and exit
```

4.13.3. Флаги контейнера Moon Basic Auth

Поддерживаются следующие флаги:

```
-f string
    htpasswd file path (default "/conf/auth")
-grace-period duration
    graceful shutdown period (default 30s)
-listen string
    address to bind (default ":4545")
-upstream-url value
    upstream url (default http://127.0.0.1:4444/)
```