**Назначение**

DigitalSkills Web Application предоставляет собой веб-сайт для организации веб-конференций. В течение двух конкурсных дней вы преобразует существующее локальное приложение в приложение на основе контейнера. Этот веб-приложение предоставит решение для размещения мультитенантных веб-приложений с использованием службы Azure Kubernetes (AKS), контейнеров Docker на узлах Linux с миграцией из MongoDB на CosmosDB.

* Службы Azure и сопутствующие продукты
* Служба Azure Kubernetes (AKS)
* Реестр контейнеров Azure
* GitHub/Azure DevOps
* Docker
* Cosmos DB (включая MongoDB API)

**Обзор**

Перед практической работой вам нужно будет подготовить среду, развернув базу данных и приложение локально на виртуальной машине с помощью Docker и MongoDB. Вам также нужно будет соединить репозиторий GitHub, содержащий лабораторию, с вашей собственной учетной записью GitHub, чтобы иметь возможность настроить конвейер CI/CD.

Требования

1. Подписка Microsoft Azure должна быть оплачиваемой по факту использования или MSDN.
2. Пробные подписки не работают.

Чтобы завершить эту лабораторную настройку, убедитесь, что в вашей учетной записи есть следующее:

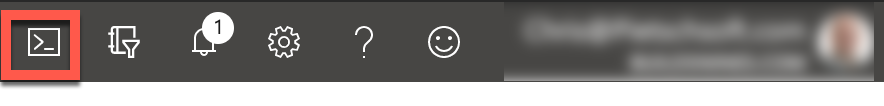
* Имеет встроенную роль владельца (Owner) для используемой вами подписки.
* Является ли пользователь-участник в используемом вами клиенте Azure AD. (У гостевых пользователей не будет необходимых разрешений.)
* В вашей подписке должно быть достаточно ядер, чтобы создать агент сборки и кластер службы Azure Kubernetes в Задаче 5: Развертывание шаблона ARM. Вам понадобится восемь ядер, если вы будете следовать точным инструкциям в лабораторной работе, и больше, если вы выберете дополнительных агентов или виртуальные машины большего размера. Выполните шаги, необходимые до начала лабораторной работы, чтобы узнать, нужно ли запрашивать дополнительные ядра в вашей подписке.
* Учетная запись в Microsoft GitHub.
* Локальная или виртуальная машина, настроенная с:
* Браузер, предпочтительно Chrome для согласованности с лабораторными тестами реализации.
* Вам будет предложено установить другие инструменты во время выполнения упражнений.

Before the hands-on lab

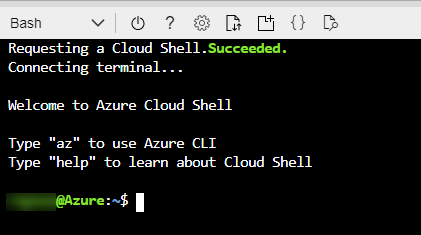
**Длительность**: 60 минут

**Задание 1: Установка Azure Cloud Shell**

1. Откройте Azure Cloud Shell, выбрав значок в строке меню..

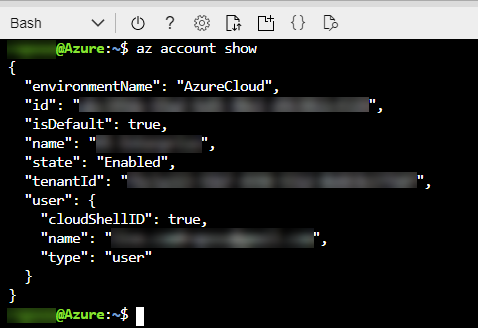


1. Azure Cloud Shell открывается в окне браузера. Выберите Bash, если будет предложено или используйте раскрывающееся меню слева в строке меню оболочки, чтобы выбрать Bash из раскрывающегося списка. При появлении запроса выберите Подтвердить.



1. Убедитесь, что подписка по умолчанию установлена правильно. Чтобы просмотреть текущий тип подписки:

az account show

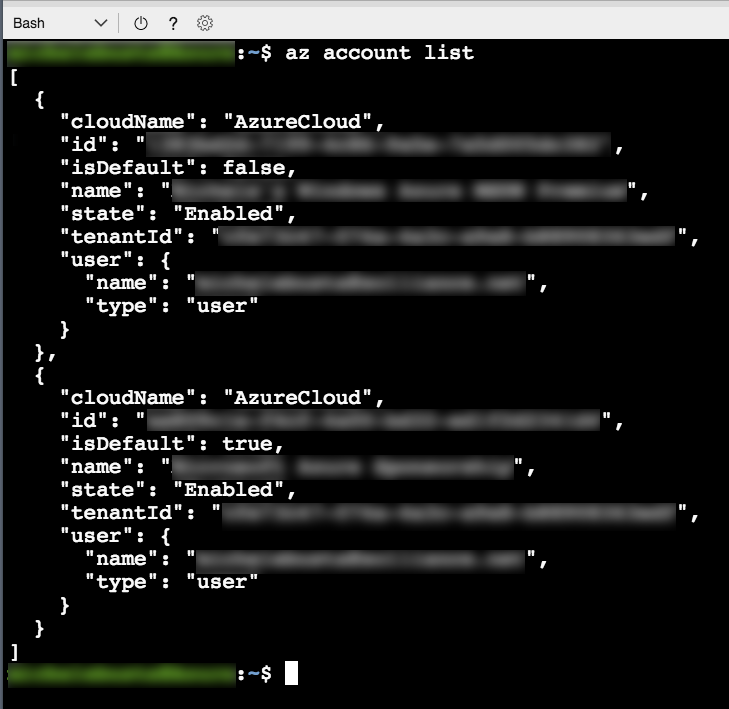
[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image37.png)

1. Чтобы установить подписку по умолчанию, отличную от текущего выбора, введите следующее, заменив {id} на желаемое значение идентификатора подписки:

az account set --subscription {id}

**Заметка**: чтобы вывести список всех ваших подписок, введите:

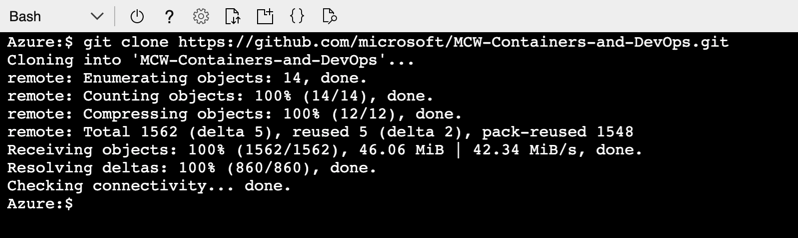
az account list

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image38.png)

**Задание 2: Получение необходимых файлов**

В этой задаче вы используете git, чтобы скопировать лабораторный контент в cloud shell.

git clone https://github.com/masyan/Cloud-applications.git

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-2019-09-30_21-25-06.png)

**Задание 3: Ресурсные группы**

Создайте группу ресурсов Azure для хранения ресурсов, которые вы создаете в этой практической лабораторной работе. Такой подход упрощает последующую очистку.

1. В окне cloud shell введите команду, аналогичную следующей, обязательно замените переменные:

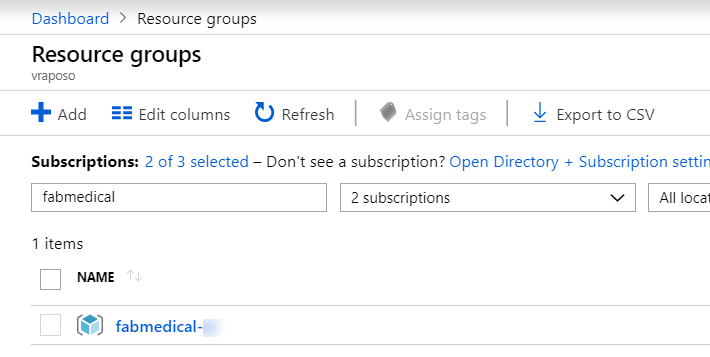
az group create -l '[LOCATION]' -n 'fabmedical-[SUFFIX]'

* + **Suffix:** На протяжении всех заданий следует использовать суффикс, чтобы сделать ресурсы уникальными, например префикс электронной почты или ваше имя и фамилию..
  + **Location:** Выберите регион для Azure Container Registry: Canada Central, Canada East, North Central US, Central US, South Central US, East US, East US 2, West US, West US 2, West Central US, France Central, UK South, UK West, North Europe, West Europe, Australia East, Australia Southeast, Brazil South, Central India, South India, Japan East, Japan West, Korea Central, Southeast Asia, East Asia и запомните это для будущих шагов, чтобы все ресурсы, которые вы создаете в Azure, хранились в одном регионе.

Например:

az group create -l 'west us' -n 'fabmedical-sol'

1. Когда этот шаг будет завершен, то на портале Azure отобразится ваша группа ресурсов.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image8.png)

**Задание 4: Создание SSH ключа**

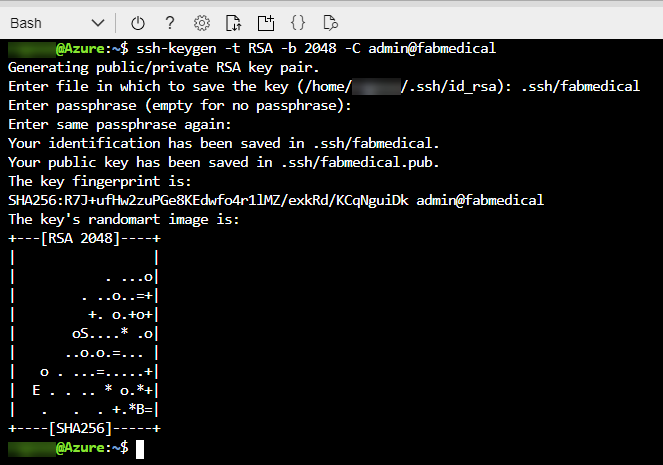
Вы создадите виртуальные машины во время следующего задания. В этом разделе вы создадите ключ SSH для безопасного доступа к виртуальным машинам.

В командной строке cloud shell введите следующую команду, чтобы убедиться, что каталог для ключей SSH существует. Вы можете игнорировать любые ошибки, которые вы видите в выводе.

mkdir .ssh

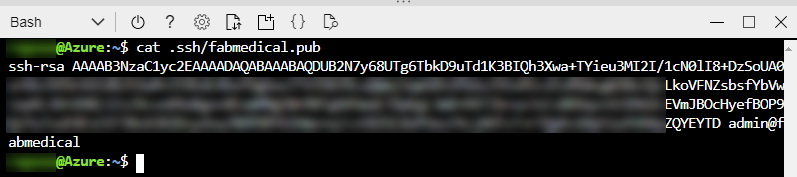
1. В командной строке cloud shell введите следующую команду, чтобы сгенерировать пару ключей SSH. Вы можете заменить admin на ваше предпочтительное имя или дескриптор.

ssh-keygen -t RSA -b 2048 -C admin@fabmedical

1. Когда вас попросят сохранить сгенерированный ключ в файл, введите в качестве имени .ssh/fabmedical
2. Введите парольную фразу и **не забудьте ее**!
3. Поскольку вы ввели .ssh/fabmedical, то ssh-keygen генерирует файл в папке .ssh в вашей пользовательской папке, где по умолчанию открывается cloud shell.[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image57.png)
4. В командной строке введите следующую команду для вывода содержимого открытого ключа. Скопируйте эту информацию, чтобы использовать позже.

cat .ssh/fabmedical.pub

1. Не закрывайте cloud shell и оставайтесь в каталоге по умолчанию. Вы будете использовать командную строку в более поздних задачах.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image571.png)

**Задача 5: Развертывание ARM шаблона**

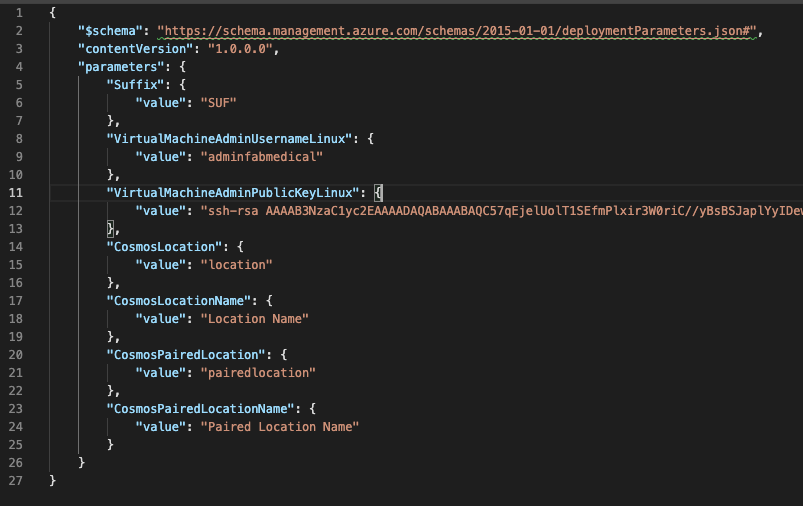
В этом разделе вы настраиваете и выполняете шаблон ARM, который создает все ресурсы, необходимые для выполнения задания.

1. Перейди в каталог с ARM шаблоном:

cd MCW-Cloud-native-applications/Hands-on\ lab/arm/

1. Откройте файл azuredeploy.parameters.json для редактирования.

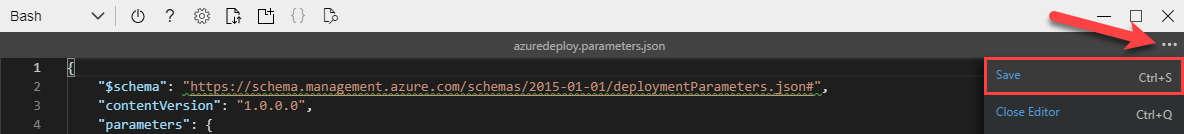
code azuredeploy.parameters.json

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image581.png)

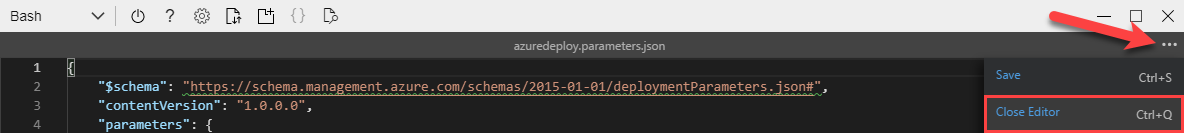
1. Обновите значения для ключей, чтобы они соответствовали вашей среде.:
   * **Suffix**: введите сокращенную версию суффикса, содержащую не более 3 символов.
   * **VirtualMachineAdminUsernameLinux**: имя пользователя администратора виртуальной машины Linux Build Agent (пример: «adminfabmedical»).
   * **VirtualMachineAdminPublicKeyLinux**: открытый ключ ssh администратора виртуальной машины Linux Build Agent. Вы найдете это значение в файле .ssh / fabmedical.pub, созданном ранее (пример: «ssh-rsa AAAAB3N (...) vPiybQV admin @ fabmedical»).
   * **CosmosLocation**: основное расположение Azure Cosmos DB. Используйте то же расположение, что и ранее созданная группа ресурсов (пример: "eastus").
   * **CosmosLocationName**: имя основного расположения Azure Cosmos DB. Используйте имя того же местоположения, что и ранее созданная группа ресурсов (пример: «eastus»).
   * **CosmosPairedLocation**: дополнительное расположение Azure Cosmos DB. Приведенную ниже ссылку можно использовать для поиска пары регионов Azure для вашего основного местоположения. (пример: "westus").
   * **CosmosPairedLocationName**: имя дополнительного расположения Azure Cosmos DB. Используйте имя местоположения, которое соответствует второстепенному местоположению, определенному в предыдущем ключе (пример: «westus»).

**Заметка**: Список регионов Azure: <https://docs.microsoft.com/en-us/azure/best-practices-availability-paired-regions#azure-regional-pairs>.

1. Выберите **...** и нажмите Сохранить.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image62.png)

1. Закройте редактор.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image63.png)

1. Создайте необходимые ресурсы с помощью ARM шаблона, заменив {resourceGroup} именем ресурсной группы, которую вы создали ранее:

az deployment group create --resource-group {resourceGroup} --template-file azuredeploy.json --parameters azuredeploy.parameters.json

Эта команда занимает от 30 до 60 минут для развертывания всех ресурсов. Вы можете перейти к следующей задаче, чтобы настроить GitHub во время развертывания.

**Задание 6: Создание GitHub репозитория**

DigitalSkills предоставила вам стартовые файлы. Это копия веб-сайта, который вам предстоит развернуть.

1. Откройте веб-браузер и перейдите на https://www.github.com. Войдите в систему, используя данные своей учетной записи GitHub.
2. В правом верхнем углу разверните раскрывающееся меню пользователя и выберите Your repositories.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-23-18-03-40.png)

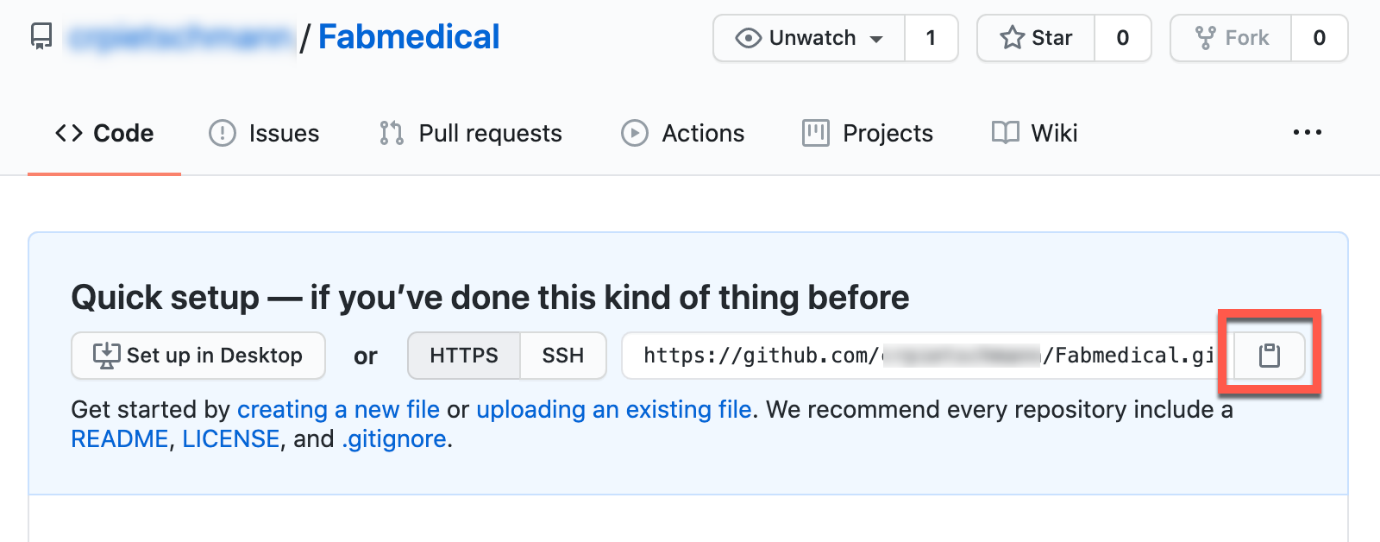
1. Рядом с поиском найдите и нажмите кнопку «Создать».

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-23-18-08-02.png)

1. На экране «Create a new repository» назовите репозиторий \*\*\*\*\*\*\*\* и нажмите кнопку «Create repository».

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-23-18-11-38.png)

1. На экране быстрой настройки скопируйте URL-адрес HTTPS GitHub для вашего нового репозитория, вставьте его в блокнот для использования в будущем.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-23-18-15-45.png)

1. Откройте новую консоль Azure Cloud Shell. Вы можете сделать это, нажав кнопку «Open new session» на первой консоли или перейдя на https://shell.azure.com и войдя в систему с теми же учетными данными лаборатории.
2. Перейдите в папку с исходным кодом FabMedical и перечислите ее содержимое.
3. cd ~/MCW-Cloud-native-applications/Hands-on\ lab/lab-files/developer/

ls

1. Вы увидите, что список включает три папки: одну для веб-сайта, другую для API контента и одну для инициализации данных API:
2. content-api/
3. content-init/

content-web/

1. Задайте свое имя пользователя и адрес электронной почты, которые git использует для коммитов.
2. git config --global user.email "you@example.com"

git config --global user.name "Your Name"

1. Используя Cloud Shell, инициализируйте новый репозиторий git:
2. git init
3. git add .

git commit -m "Initial Commit"

1. Установите для удаленного источника URL-адрес GitHub, введя следующую команду:

git remote add origin <your GitHub URL>

1. Настройте git CLI для кэширования ваших учетных данных, чтобы вам не приходилось повторно вводить их.
2. git config --global --unset credential.helper

git config --global credential.helper store

1. Перейдите в основную ветку, введя следующую команд:
2. git branch -m master main

git push -u origin main

1. Обновите репозиторий GitHub, теперь вы должны увидеть опубликованный код.

**Задание 7: Подключение к build агенту**

В этом разделе вы проверим, что сможем подключиться к новой виртуальной машине с агентом для сборки.

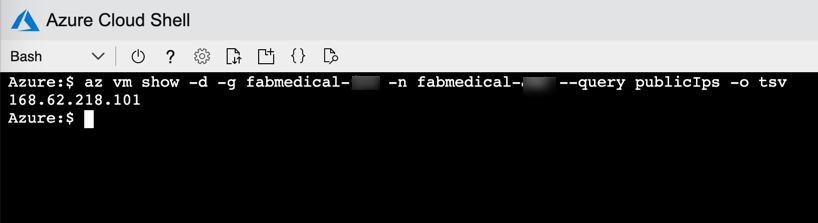
1. Откройте новую консоль Azure Cloud Shell и выполните следующую команду, чтобы найти IP-адрес виртуальной машины агента сборки, подготовленной при запуске развертывания ARM.:

az vm show -d -g fabmedical-[SUFFIX] -n fabmedical-[SHORT\_SUFFIX] --query publicIps -o tsv

Например:

az vm show -d -g fabmedical-sol -n fabmedical-SOL --query publicIps -o tsv

1. В ответе обратите внимание на общедоступный IP-адрес виртуальной машины..

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-2019-10-01_11-58-05.png)

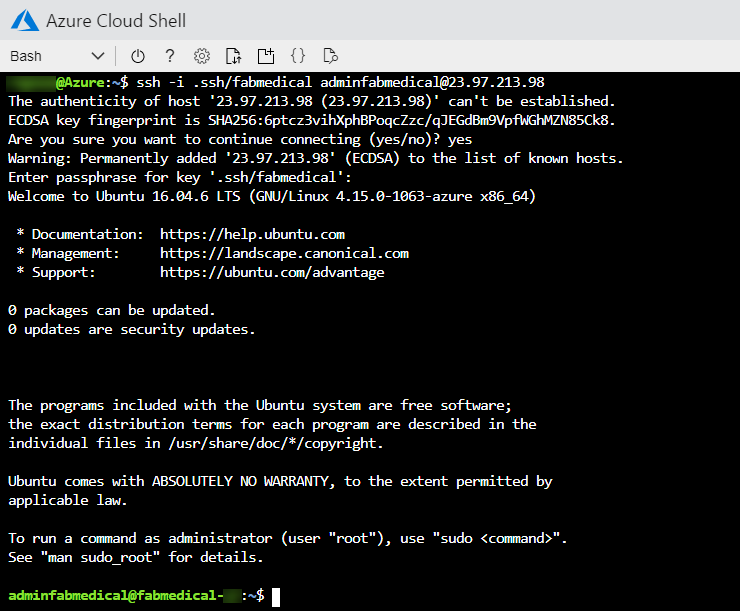
1. Подключитесь к новой виртуальной машине, которую вы создали, введя следующую команду:

ssh -i [PRIVATEKEYNAME] [BUILDAGENTUSERNAME]@[BUILDAGENTIP]

ssh -i .ssh/fabmedical adminfabmedical@52.174.141.11

1. Когда вас попросят подтвердить, хотите ли вы подключиться, поскольку подлинность подключения не может быть подтверждена, введите «да».
2. Когда вас попросят ввести кодовую фразу для ранее созданного закрытого ключа, введите это значение.
3. SSH подключается к виртуальной машине и отображает следующую командную строку. Оставьте это окно облачной оболочки открытым для следующего шага:

adminfabmedical@fabmedical-SUFFIX:~$

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image27.png)

**Задание 8: Завершите конфигурацию build агента**

В этой задаче вы обновляете пакеты и устанавливаете Docker.

1. Перейдите в окно Cloud Shell, в котором открыто SSH-соединение с виртуальной машиной агента сборки.
2. Обновите пакеты Ubuntu и установите curl и поддержку репозиториев через HTTPS за один шаг, набрав следующую команду в одной строке. Ответьте, набрав Y и нажав клавишу ВВОД, если вас спросят, хотите ли вы продолжить.

sudo apt-get update && sudo apt install apt-transport-https ca-certificates curl software-properties-common

1. Добавьте официальный ключ GPG Docker, введя в однострочную команду следующее::

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

1. Добавьте стабильный репозиторий Docker в список пакетов Ubuntu, набрав в однострочную команду следующую команду:

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

1. Добавьте NodeJs PPA, чтобы использовать выпуск NodeJS LTS, обновить пакеты Ubuntu и установить механизм Docker, node.js и npm, введя следующие команды, каждую в отдельной строке. Если вас спросят, хотите ли вы продолжить, ответьте, набрав Y и нажав Enter.
2. sudo apt-get install curl python-software-properties -y
3. curl -sL https://deb.nodesource.com/setup\_12.x | sudo -E bash -

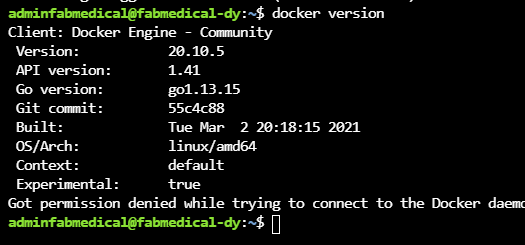
sudo apt-get update && sudo apt-get install -y docker-ce nodejs mongodb-clients

1. Теперь обновите пакеты Ubuntu до последней версии, введя в однострочную команду следующее:.

sudo apt-get upgrade -y

1. Когда команда будет завершена, проверьте установленную версию Docker, выполнив эту команду. Результат может выглядеть примерно так, как показано на следующем снимке экрана. Обратите внимание, что версия сервера еще не отображается, потому что вы не запускали команду с повышенными привилегиями (мы рассмотрим это в ближайшее время).

docker version

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/docker-version.png)

1. Вы также можете проверить версии node.js и npm, просто для информации, используя эти команды:
2. nodejs --version

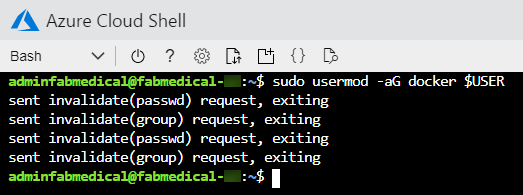
npm -version

1. Установите Angular CLI.

sudo npm install -g @angular/cli

1. Чтобы удалить требование использовать sudo, добавьте своего пользователя в группу Docker. Вы можете игнорировать любые ошибки, которые видите в выводе.

sudo usermod -aG docker $USER

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image29.png)

1. Чтобы изменения прав пользователя вступили в силу, выйдите из сеанса SSH, набрав exit, затем нажмите <Enter>. Повторно подключитесь к виртуальной машине агента сборки с помощью SSH, как вы это делали в предыдущей задаче.

ssh -i .ssh/fabmedical adminfabmedical@52.174.141.11

1. Повторите команду docker version и обратите внимание, что в выводе теперь также отображается версия сервера.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/docker-version-server.png)

1. Выполните несколько команд Docker:

docker container ls

docker container ls -a

1. В обоих случаях у вас есть пустой список, но нет ошибок при выполнении команды. Ваш агент сборки готов, и движок Docker работает правильно.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b4-image31.png)

**Задание 9: Склонируйте репозиторий на Build агента**

В этой задаче вы клонируете свои репозитории из GitHub, чтобы вы могли работать с ними в агенте сборки.

1. Как и ранее в облачной оболочке, укажите имя пользователя и адрес электронной почты, которые используются для коммитов git.
2. git config --global user.email "you@example.com"

git config --global user.name "Your Name"

**Note**: В некоторых случаях пользователь root владеет папкой .config вашего пользователя. Если это произойдет, выполните следующую команду, чтобы вернуть право собственности на adminfabmedical, а затем попробуйте команду git еще раз.:

sudo chown -R $USER:$(id -gn $USER) /home/adminfabmedical/.config

1. Настройте git CLI для кеширования учетных данных, чтобы вам не приходилось повторно вводить их.

git config --global credential.helper cache

1. Используйте URL-адрес GitHub, чтобы клонировать код репозитория на свой компьютер с агентом сборки.

git clone <GITHUB\_REPOSITORY\_URL>

**Задание 10: Запустите стартовое приложение**

В этой задаче вы возьмете стартовые файлы и запустите приложение node.js как приложение Docker. Вы создадите образы Docker из существующих файлов и запустите контейнеры для тестирования и выполнения приложения.

1. Из Azure Cloud Shell подключитесь к своему агенту сборки, если вы еще не подключены.
2. Введите следующую команду, чтобы создать сеть Docker с именем fabmedical:

docker network create fabmedical

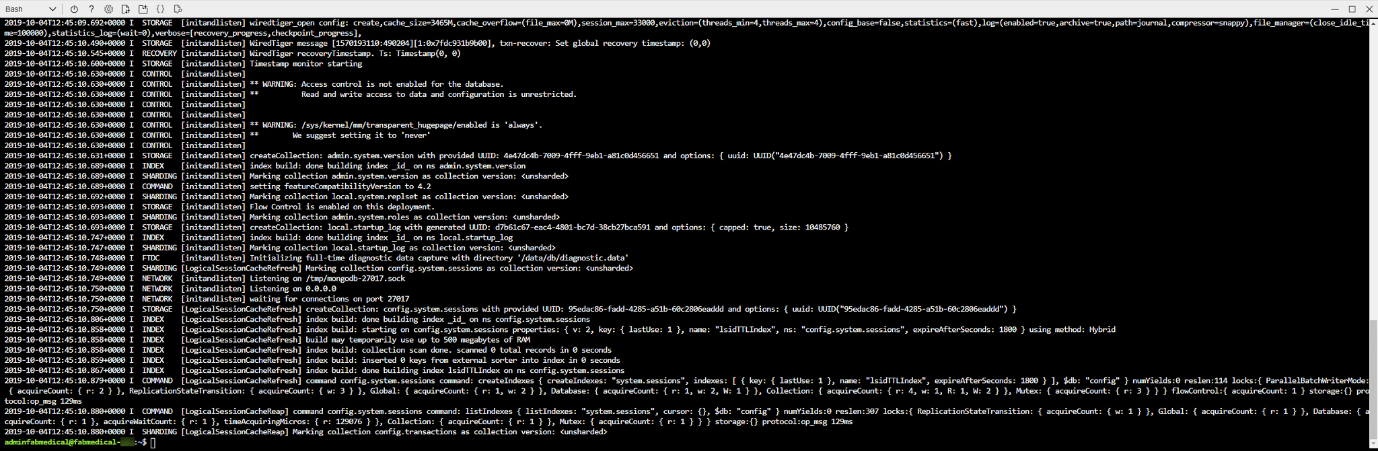
1. Запустите экземпляр mongodb, чтобы использовать его для локального тестирования.

docker container run --name mongo --net fabmedical -p 27017:27017 -d mongo:4.0

**Note**: С существующим исходным кодом, написанным для MongoDB, его можно указать на конечную точку API MongoDB в Azure Cosmos DB. Эмулятор Azure Cosmos DB можно использовать для локальной разработки в Windows; однако эмулятор Cosmos DB не поддерживает Linux. В результате при использовании Linux для разработки MongoDB по-прежнему требуется для локальных сред разработки; с использованием Azure Cosmos DB для хранения данных в облаке. Это позволяет легко перенести существующий исходный код, написанный для хранилища MongoDB, на серверную часть Azure Cosmos DB.

1. Убедитесь, что контейнер mongo запущен и готов.
2. docker container list

docker container logs mongo

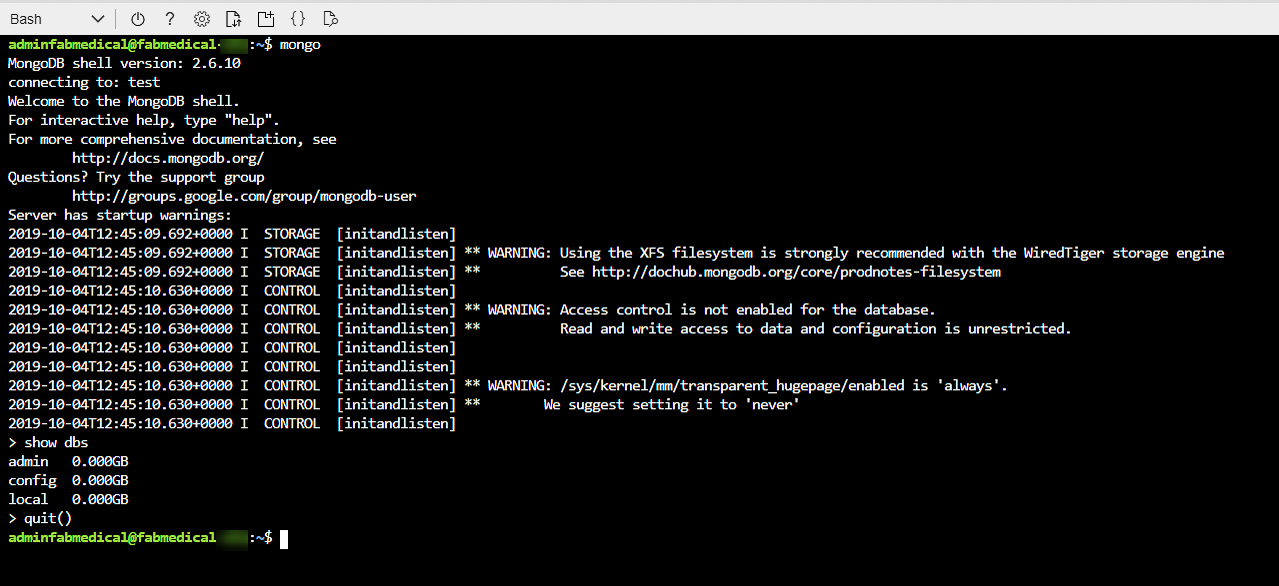
[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b410-Step4.1.png)

1. Подключитесь к экземпляру mongo с помощью оболочки mongo и проверьте некоторые основные команды:

mongo

show dbs

quit()

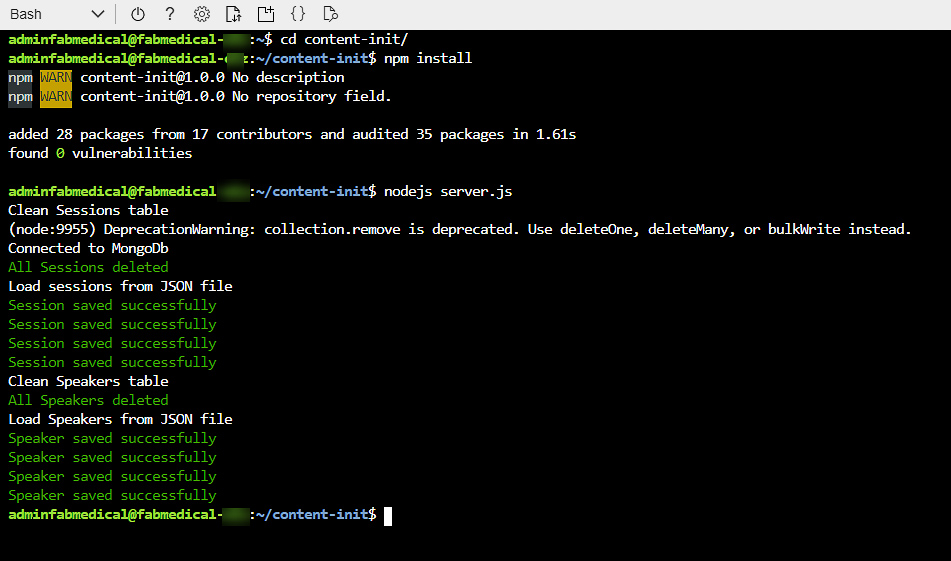
[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b410-Step5.1.png)

1. Чтобы инициализировать локальную базу данных тестовым содержимым, сначала перейдите в каталог content-init и запустите npm install.
2. cd ~/Fabmedical/content-init

npm install

1. Инициализируйте базу данных.

nodejs server.js

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b410-Step7.1.png)

1. Убедитесь, что база данных теперь содержит тестовые данные.

mongo

show dbs

use contentdb

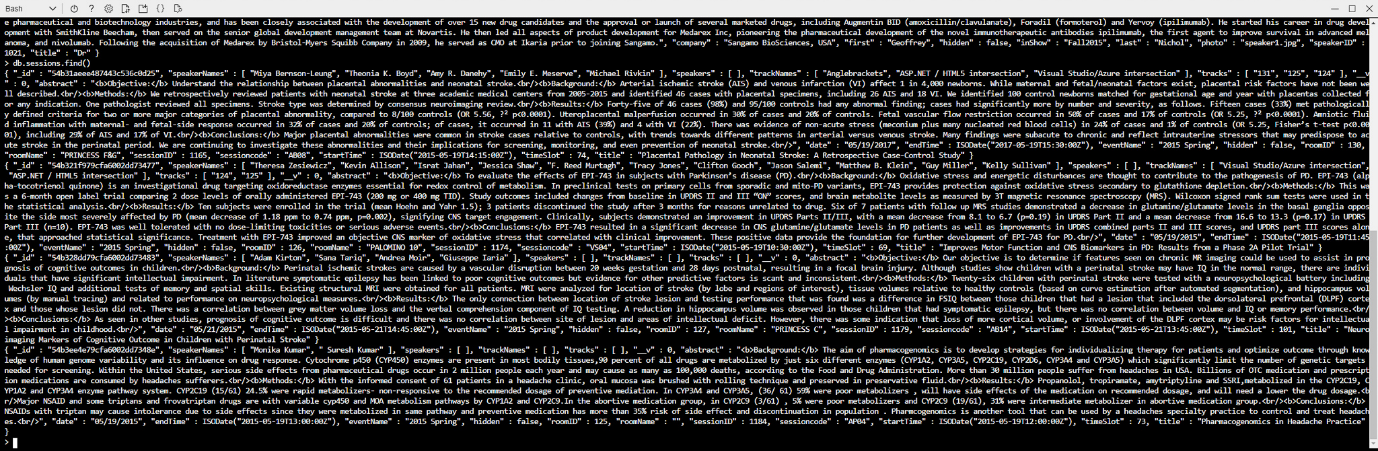
show collections

db.speakers.find()

db.sessions.find()

quit()

Это должно привести к выводу, подобному следующему:

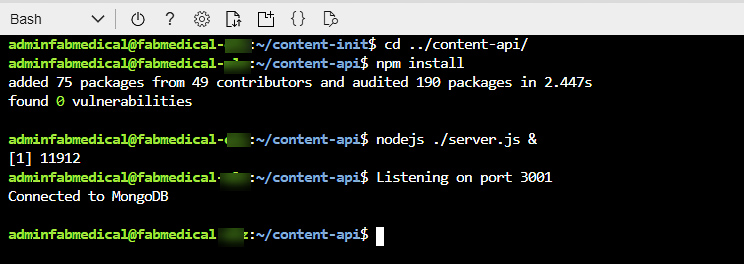
[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/b410-Step8.1.png)

1. Теперь перейдите в каталог content-api и запустите npm install.
2. cd ../content-api

npm install

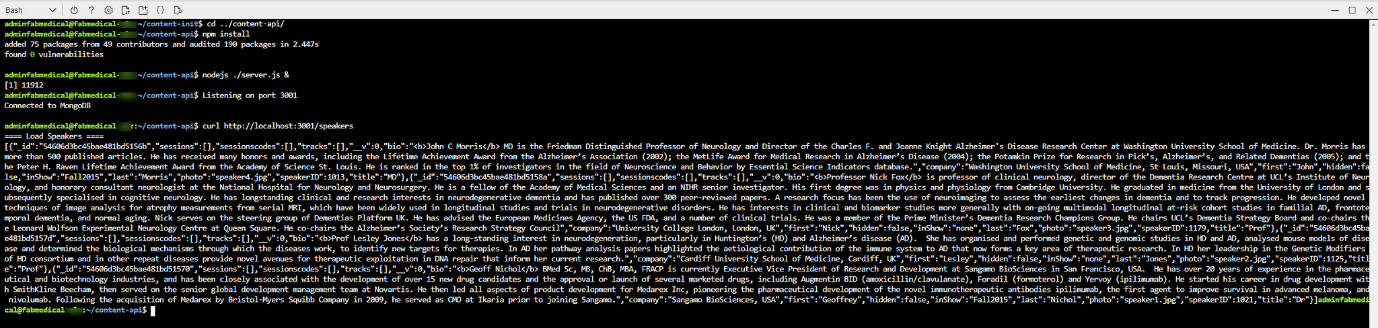
1. Запустите API в фоновом режиме.

nodejs ./server.js &

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image47.png)

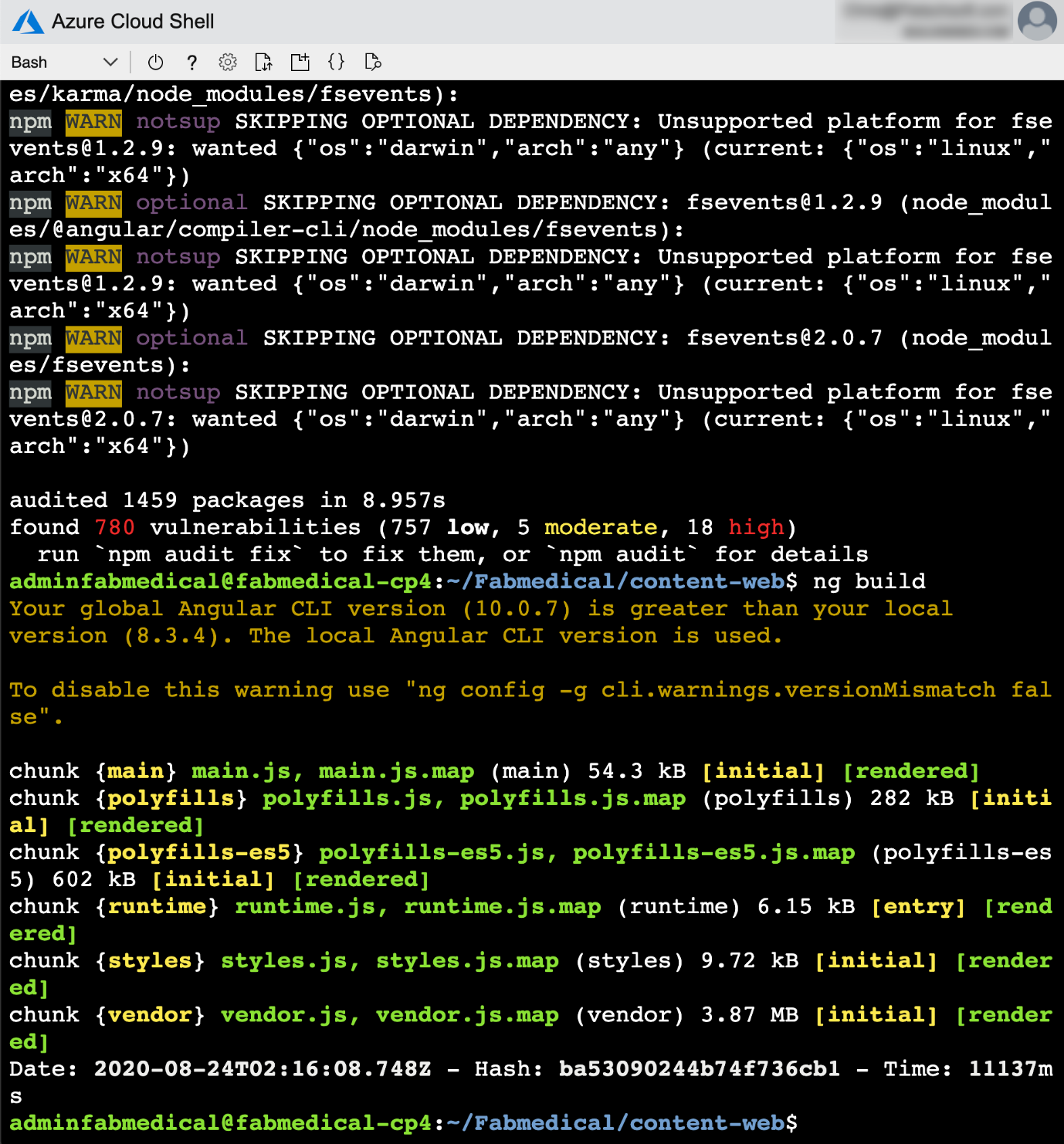
1. Снова нажмите клавишу Enter, чтобы перейти к командной строке для следующего шага.
2. Протестируйте API с помощью curl. Вы запросите контент докладчика, и он вернет результат в формате JSON.

curl http://localhost:3001/speakers

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image47_1.png)

1. Перейдите в каталог веб-приложения, запустите npm install и ng build.
2. cd ../content-web
3. npm install

ng build

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image48.png)

1. В Azure Сloud Shell выполните следующую команду, чтобы найти IP-адрес виртуальной машины агента сборки, подготовленной при запуске развертывания ARM.

az vm show -d -g fabmedical-[SUFFIX] -n fabmedical-[SHORT\_SUFFIX] --query publicIps -o tsv

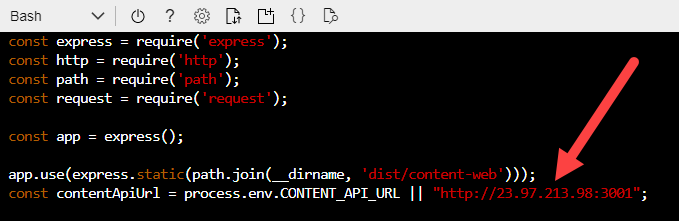
Например:

az vm show -d -g fabmedical-sol -n fabmedical-SOL --query publicIps -o tsv

1. Из облачной оболочки на машине сборки отредактируйте файл app.js с помощью vim.

vim app.js

Затем нажмите i, чтобы перейти в режим редактирования, после чего замените localhost на IP-адрес машины сборки.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image27.png)

Затем нажмите ESC, напишите: wq, чтобы сохранить изменения и закрыть файл.

1. Теперь запустите контент-веб-приложение в фоновом режиме.

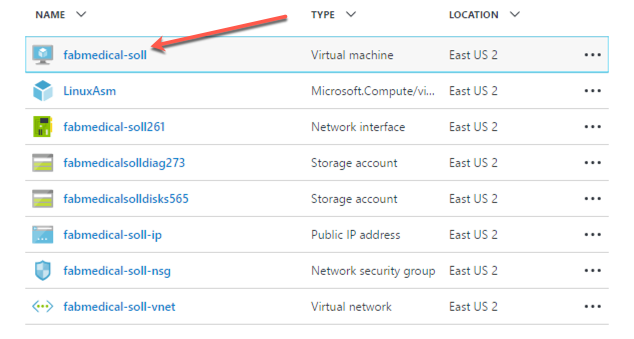
node ./app.js &

Снова нажмите клавишу ВВОД, чтобы получить командную строку для следующего шага.

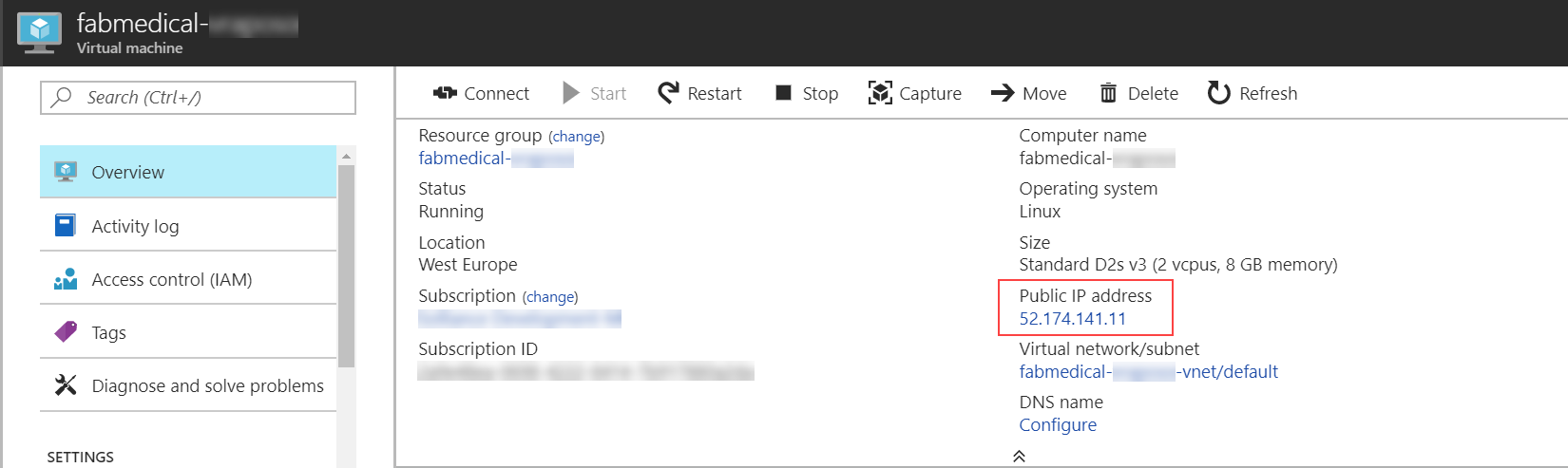
1. Протестируйте веб-приложение с помощью curl. Вы увидите, что вывод HTML возвращается без ошибок.

curl http://localhost:3000

1. Оставьте приложение запущенным для выполнения следующей задачи.
2. Если вы получили ответ JSON на запрос содержимого / динамиков и ответ HTML от веб-приложения, ваша среда работает должным образом.
3. На портале Azure выберите созданную вами группу ресурсов с именем fabmedical-SUFFIX.
4. Выберите виртуальную машину агента сборки с именем fabmedical-SUFFIX из списка доступных ресурсов.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image54.png)

1. В обзоре виртуальной машины найдите IP-адрес виртуальной машины.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image26.png)

1. Протестируйте веб-приложение в браузере. Перейдите к веб-приложению, используя IP-адрес вашего агента сборки на порту 3000.

http://[BUILDAGENTIP]:3000

EXAMPLE: http://13.68.113.176:3000

1. Выберите ссылки «Спикеры» и «Сеансы» в заголовке. Вы увидите, что страницы отображают HTML-версию содержимого JSON, которое вы скручивали ранее.
2. Убедившись, что приложение доступно через браузер, перейдите в окно облачной оболочки и остановите запущенные процессы узла.

killall nodejs

killall node

**Задание 11: Сборка Docker образов**

В этой задаче вы создадите образы Docker для приложения - один для приложения API, а другой - для веб-приложения. Каждый образ будет создан с помощью команд Docker, которые полагаются на Dockerfile.

1. В Cloud Shell, подключенном к виртуальной машине агента сборки, введите следующую команду, чтобы просмотреть любые образы Docker на виртуальной машине. В списке будет только образ mongodb, загруженный ранее.

docker image ls

1. В папке content-api (cd ~ / Fabmedical / content-api), содержащей файлы приложения API и файл Dockerfile, введите следующую команду, чтобы создать образ Docker для приложения API. Эта команда выполняет следующие действия:
   * Выполняет команду сборки Docker для создания образа
   * Помечает получившееся изображение с именем content-api (-t)
   * Последняя точка (.) указывает на использование Dockerfile в контексте текущего каталога. По умолчанию ожидается, что этот файл будет иметь имя Dockerfile (с учетом регистра).

docker image build -t content-api .

1. После успешного создания образа снова запустите команду Docker images list. Вы увидите несколько новых изображений: изображения узлов и изображение вашего контейнера.

docker image ls

Обратите внимание на немаркированное изображение. Это этап сборки, который содержит все промежуточные файлы, которые не нужны в вашем окончательном образе.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image59.png)

1. Снова перейдите в папку content-web и выведите список файлов. Обратите внимание, что в этой папке также есть Dockerfile.
2. cd ../content-web

ll

1. Просмотрите содержимое Dockerfile. Введите следующую команду:

cat Dockerfile

Обратите внимание, что этап сборки файла Dockerfile с веб-контентом включает в себя дополнительные инструменты для интерфейсного приложения Angular в дополнение к установке пакетов npm.

1. Введите следующую команду, чтобы создать образ Docker для веб-приложения.

docker image build -t content-web .

1. Снова перейдите в папку content-init и перечислите файлы. Обратите внимание, что в этой папке уже есть Dockerfile.
2. cd ../content-init

ll

1. Просмотрите содержимое Dockerfile. Введите следующую команду:

cat Dockerfile

1. Введите следующую команду, чтобы создать образ Docker для приложения инициализации.

docker image build -t content-init .

1. По завершении вы увидите восемь образов, которые теперь существуют, когда вы запустите команду Docker images.

docker image ls

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/vm-list-containers.PNG)

**Задание 12: Запуск Docker контейнеров**

Контейнер веб-приложения будет вызывать конечные точки, предоставляемые контейнером приложения API, а контейнер приложения API будет взаимодействовать с mongodb. В этом упражнении вы запустите созданные вами образы как контейнеры в той же сети моста, которую вы создали при запуске mongodb.

1. Создайте и запустите контейнер приложения API с помощью следующей команды. Команда делает следующее:
   * Называет контейнер api для дальнейшего использования с командами Docker.
   * Дает указание движку Docker использовать сеть fabmedical.
   * Указывает подсистеме Docker использовать порт 3001 и сопоставить его с внутренним портом контейнера 3001.
   * Создает контейнер из указанного изображения по его тегу, например content-api.

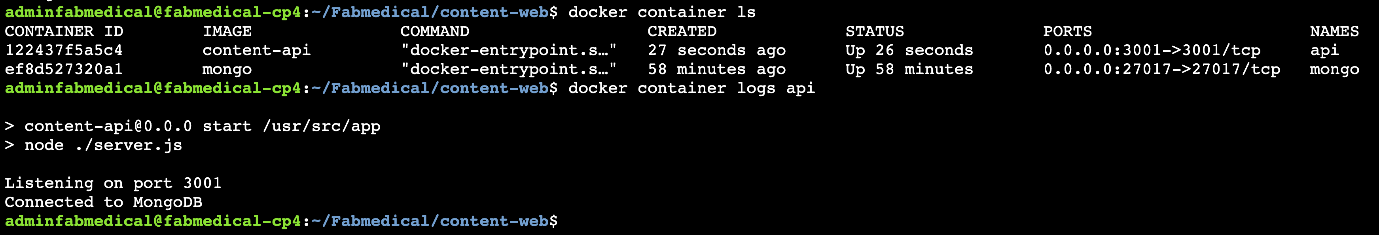
docker container run --name api --net fabmedical -p 3001:3001 content-api

1. Команда запуска контейнера докеров не удалась, поскольку она настроена для подключения к mongodb с использованием URL-адреса localhost. Однако теперь, когда content-api изолирован в отдельном контейнере, он не может получить доступ к mongodb через localhost даже при работе на том же хосте докеров. Вместо этого API должен использовать мостовую сеть для подключения к mongodb.
2. > content-api@0.0.0 start
3. > node ./server.js
4. Listening on port 3001
5. Could not connect to MongoDB!
6. MongooseServerSelectionError: connect ECONNREFUSED 127.0.0.1:27017
7. npm notice
8. npm notice New patch version of npm available! 7.0.8 -> 7.0.13
9. npm notice Changelog: <https://github.com/npm/cli/releases/tag/v7.0.13>
10. npm notice Run `npm install -g npm@7.0.13` to update!
11. npm notice
12. npm ERR! code 255
13. npm ERR! path /usr/src/app
14. npm ERR! command failed
15. npm ERR! command sh -c node ./server.js
16. npm ERR! A complete log of this run can be found in:
17. npm ERR! /root/.npm/\_logs/2020-11-23T03\_04\_12\_948Z-debug.log
18. Приложение content-api позволяет переменной среды настраивать строку подключения mongodb. Удалите существующий контейнер, а затем проинструктируйте движок докеров установить переменную среды, добавив переключатель -e в команду запуска контейнера докеров. Кроме того, используйте переключатель -d для запуска api как демона.
19. docker container rm api

docker container run --name api --net fabmedical -p 3001:3001 -e MONGODB\_CONNECTION=mongodb://mongo:27017/contentdb -d content-api

1. Введите команду, чтобы показать запущенные контейнеры. Вы заметите, что контейнер api находится в списке. Используйте команду docker logs, чтобы увидеть, что приложение API подключилось к mongodb.
2. docker container ls

docker container logs api

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image61.png)

1. Протестируйте API, свернув URL-адрес. Вы увидите вывод JSON, как и при предыдущем тестировании.

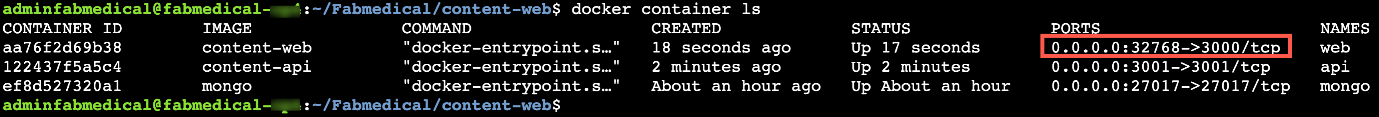
curl http://localhost:3001/speakers

1. Создайте и запустите контейнер веб-приложения с помощью аналогичной команды запуска контейнера докера - укажите движку докера использовать любой порт с помощью команды -P.

docker container run --name web --net fabmedical -P -d content-web

1. Введите команду, чтобы снова показать запущенные контейнеры, и вы увидите, что в списке есть и API, и веб-контейнеры. Веб-контейнер показывает динамически назначаемое сопоставление порта с внутренним портом контейнера 3000.

docker container ls

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image62.png)

1. Протестируйте веб-приложение, получив URL-адрес с помощью curl. Для порта используйте динамически назначаемый порт, который вы можете найти в выходных данных предыдущей команды. Вы увидите вывод HTML, как и при предыдущем тестировании.

curl http://localhost:[PORT]/speakers.html

**Задание 13: Конфигурация переменных окружения**

В этой задаче вы настроите веб-контейнер для связи с контейнером API с помощью переменной среды, аналогично тому, как строка подключения mongodb предоставляется API-интерфейсу.

1. В Cloud Shell, подключенном к виртуальной машине агента сборки, остановите и удалите веб-контейнер, используя следующие команды.
2. docker container stop web

docker container rm web

1. Убедитесь, что веб-контейнер больше не работает или не присутствует, используя флаг -a, как показано в этой команде. Вы увидите, что веб-контейнера больше нет в списке.

docker container ls -a

1. Проверьте app.js
2. cd ../content-web

cat app.js

1. Обратите внимание, что переменная contentApiUrl может быть установлена с помощью переменной среды.

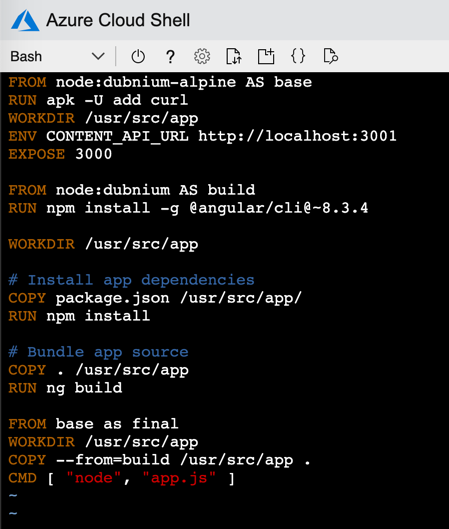
const contentApiUrl = process.env.CONTENT\_API\_URL || "http://[VM IP]:3001";

1. Откройте Dockerfile для редактирования с помощью Vim и нажмите клавишу i, чтобы перейти в режим редактирования.
2. vi Dockerfile

<i>

1. Найдите строку EXPOSE, показанную ниже, и добавьте строку над ней, которая устанавливает значение по умолчанию для переменной среды, как показано на снимке экрана.

ENV CONTENT\_API\_URL http://localhost:3001

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/hol-2019-10-01_19-37-35.png)

1. Восстановите образ Docker веб-приложения, используя ту же команду, что и раньше.

docker image build -t content-web .

1. Создайте и запустите образ, передав правильный URI в контейнер API в качестве переменной среды. Эта переменная будет обращаться к приложению API, используя его имя контейнера в созданной вами сети Docker. После запуска контейнера убедитесь, что он запущен, и обратите внимание на динамическое назначение порта для следующего шага.
2. docker container run --name web --net fabmedical -P -d -e CONTENT\_API\_URL=http://api:3001 content-web

docker container ls

1. Снова выполните команду curl, используя порт, назначенный веб-контейнеру. Опять же, вы увидите возвращенный HTML, но поскольку curl не обрабатывает javascript, вы не можете определить, взаимодействует ли веб-приложение с приложением api. Вы должны проверить это соединение в браузере.

curl http://localhost:[PORT]/speakers.html

1. Вы не сможете просматривать веб-приложение на временном порте, потому что виртуальная машина предоставляет только ограниченный диапазон портов. Теперь вы остановите веб-контейнер и перезапустите его, используя порт 3000 для тестирования в браузере. Введите следующие команды, чтобы остановить контейнер, удалить его и снова запустить, используя явные настройки для порта.
2. docker container stop web
3. docker container rm web

docker container run --name web --net fabmedical -p 3000:3000 -d -e CONTENT\_API\_URL=http://api:3001 content-web

1. Используйте Curl снова на порту 3000. Вы увидите тот же HTML-код.

curl http://localhost:3000/speakers.html

1. Теперь вы можете использовать веб-браузер для перехода на веб-сайт и успешного просмотра приложения через порт 3000. Замените [BUILDAGENTIP] на IP-адрес, который вы использовали ранее.
2. http://[BUILDAGENTIP]:3000

EXAMPLE: http://13.68.113.176:3000

1. Зафиксируйте свои изменения и отправьте в репозиторий.
2. cd ~/Fabmedical
3. git add .
4. git commit -m "Setup Environment Variables"

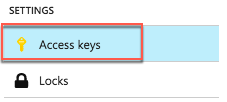
git push

**Задание 14: Отправка образов в Azure Container Registry**

Для запуска контейнеров в удаленной среде вы обычно отправляете образы в реестр Docker, где вы можете хранить и распространять образы. У каждой службы будет репозиторий, который можно отправлять и извлекать с помощью команд Docker. Реестр контейнеров Azure (ACR) - это управляемая частная служба реестра Docker, основанная на реестре Docker v2.

В этой задаче вы отправите образы в свою учетную запись ACR, образы версий с тегами и настроите непрерывную интеграцию (CI) для создания будущих версий ваших контейнеров и автоматически отправите их в ACR.

1. На портале Azure перейдите к ACR, который вы создали перед практической лабораторной работой.
2. Выебрите **Access keys** в разделе **Settings**.

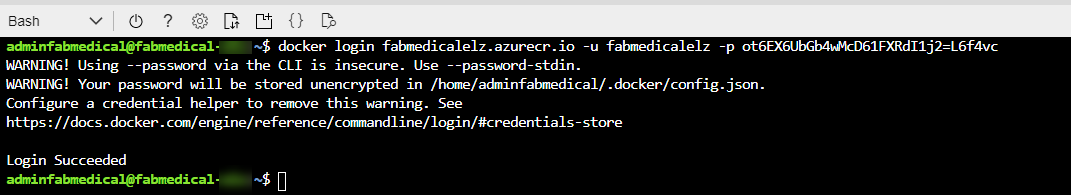
[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image64.png)

1. В сеансе облачной оболочки, подключенном к вашей виртуальной машине сборки, войдите в свою учетную запись ACR, введя следующую команду. Следуйте инструкциям для завершения входа в систему.

docker login [LOGINSERVER] -u [USERNAME] -p [PASSWORD]

Например:

docker login fabmedicalsoll.azurecr.io -u fabmedicalsoll -p +W/j=l+Fcze=n07SchxvGSlvsLRh/7ga

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image65.png)

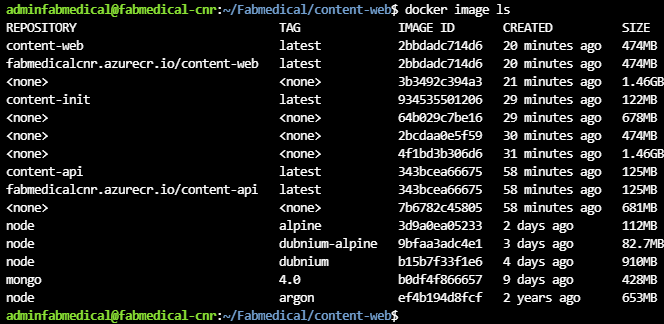
1. Выполните следующие команды, чтобы правильно пометить изображения, чтобы они соответствовали имени вашей учетной записи ACR.
2. docker image tag content-web [LOGINSERVER]/content-web

docker image tag content-api [LOGINSERVER]/content-api

**Note**: Be sure to replace the [LOGINSERVER] of your ACR instance.

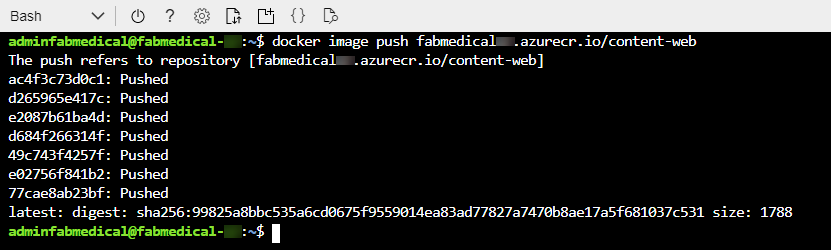
1. Перечислите свои образы докеров и посмотрите репозиторий и тег. Обратите внимание, что к репозиторию добавляется префикс вашего имени сервера входа в ACR, как в примере, показанном на снимке экрана ниже.

docker image ls

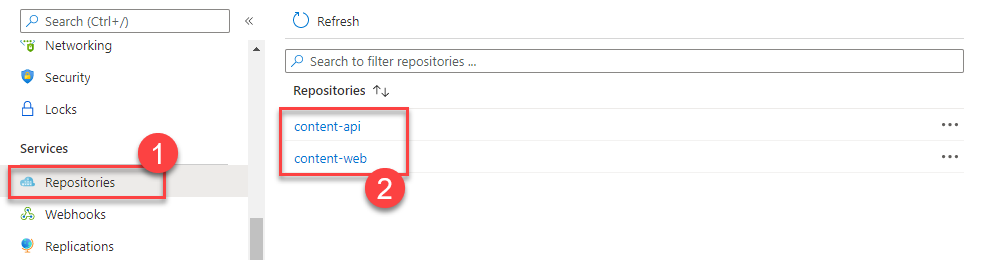
[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/tagged-images-for-acr.png)

1. Отправьте образы в свою учетную запись ACR с помощью следующей команды:
2. docker image push [LOGINSERVER]/content-web

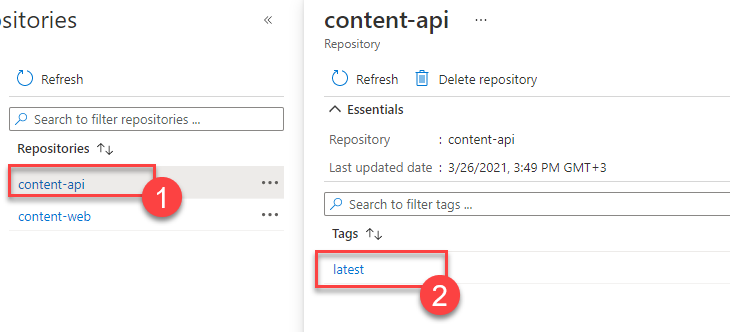
docker image push [LOGINSERVER]/content-api

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image67.png)

1. На портале Azure перейдите к своей учетной записи ACR и выберите «Репозитории» в разделе «Службы» в меню слева. Теперь вы увидите два контейнера (2), по одному для каждого изображения.

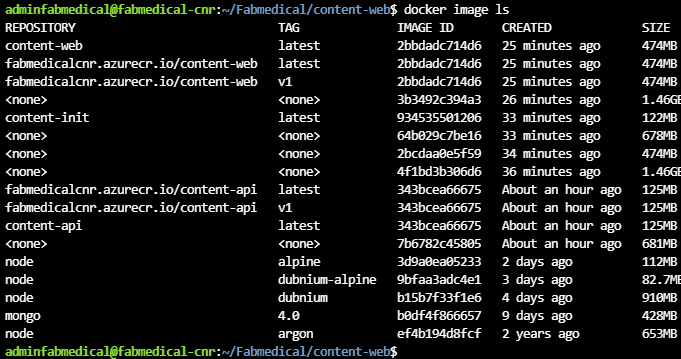
[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/acr-two-containers.png)

1. Выберите content-api (1). Вы увидите, что последний тег (2) назначен.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/acr-content-api-latest.png)

1. В сеансе облачной оболочки, подключенном к виртуальной машине, назначьте тег v1 каждому изображению с помощью следующих команд. Затем перечислите образы Docker, чтобы отметить, что теперь для каждого образа есть две записи: с указанием последнего тега latest и тега v1. Также обратите внимание, что ID изображения одинаков для двух записей, так как имеется только одна копия изображения.
2. docker image tag [LOGINSERVER]/content-web:latest [LOGINSERVER]/content-web:v1
3. docker image tag [LOGINSERVER]/content-api:latest [LOGINSERVER]/content-api:v1

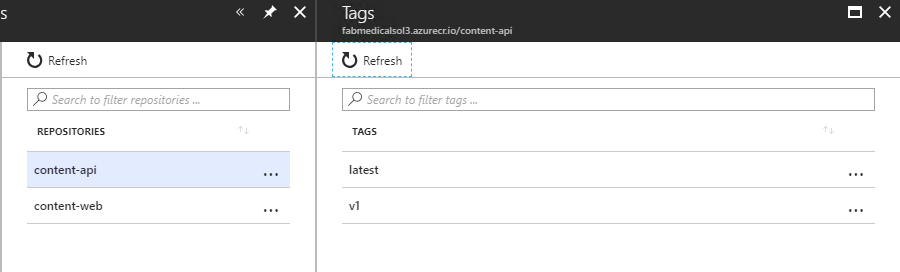
docker image ls

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/newly-tagged-images.png)

1. Отправьте образы в свою учетную запись ACR с помощью следующей команды:
2. docker image push [LOGINSERVER]/content-web:v1

docker image push [LOGINSERVER]/content-api:v1

1. Обновите один из репозиториев, чтобы теперь появились две версии изображения.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/image71.png)

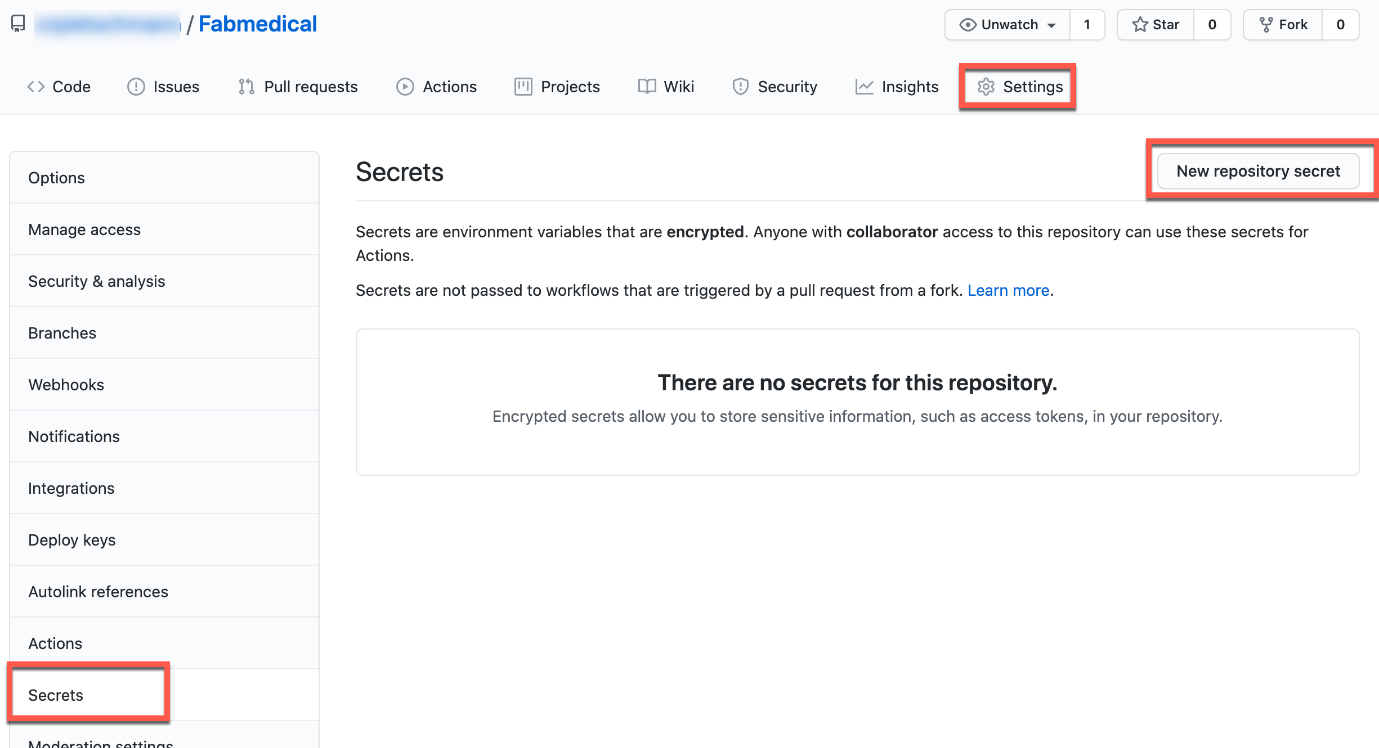
1. Выполните следующие команды, чтобы получить изображение из репозитория. Обратите внимание, что по умолчанию извлекаются изображения с тегами latest. Вы можете получить конкретную версию с помощью тега версии. Также обратите внимание, что, поскольку изображения уже существуют в агенте сборки, ничего не загружается.
2. docker image pull [LOGINSERVER]/content-web

docker image pull [LOGINSERVER]/content-web:v1

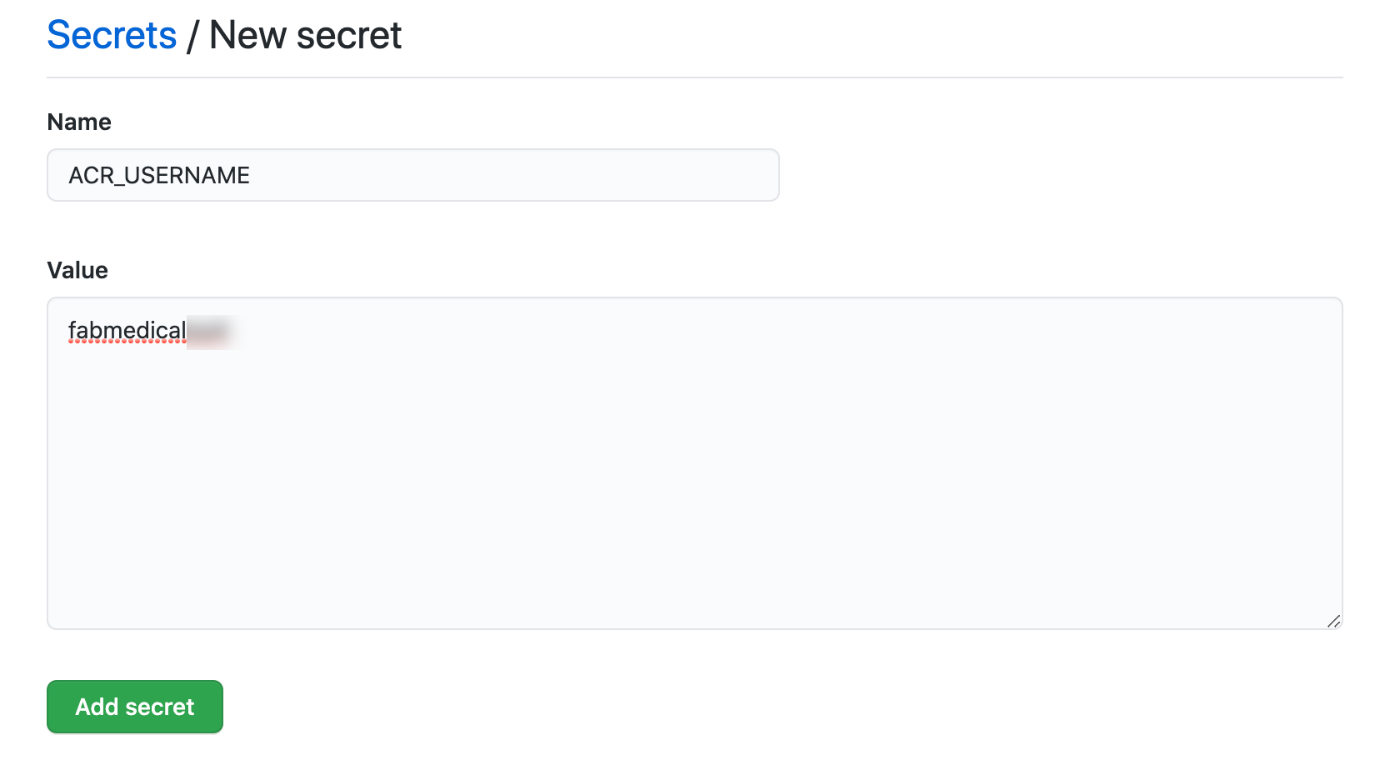
**Задание 15: Настройка CI Pipeline для отправки образов**

В этой задаче вы будете использовать YAML для определения рабочего процесса GitHub Actions, который создает ваш образ Docker и автоматически отправляет его в ваш экземпляр ACR.

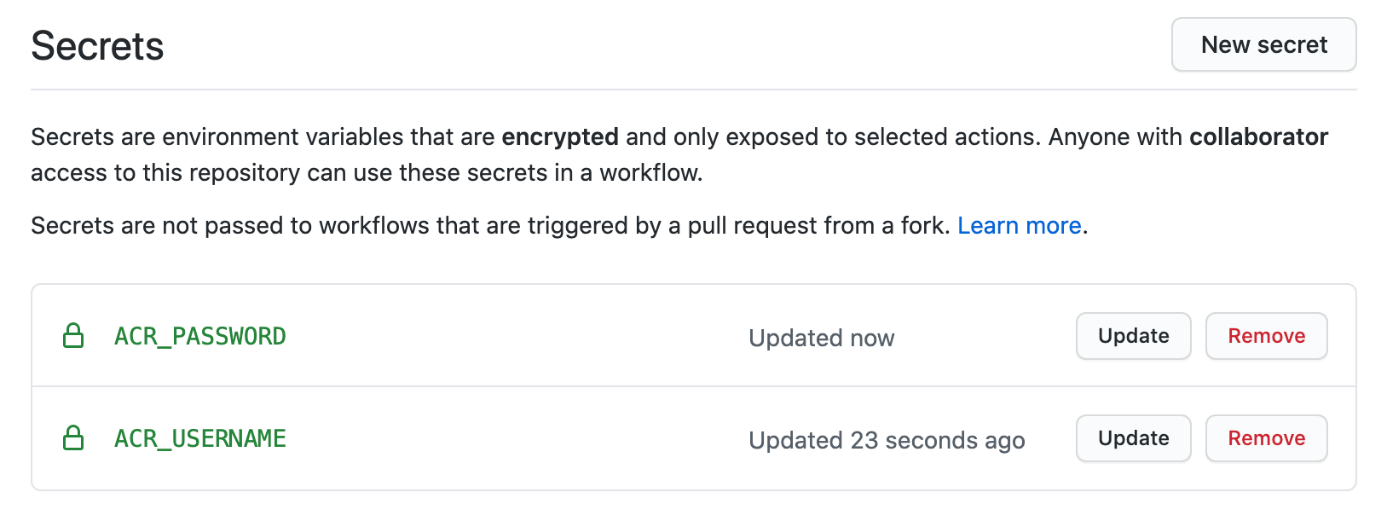
1. В GitHub вернитесь на экран репозитория и выберите вкладку «Настройки»..
2. В меню слева выберите "Секреты".
3. Нажмите кнопку «Новый секрет репозитория».

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-24-21-45-42.png)

1. В форме «Новый секрет» введите имя ACR\_USERNAME и в качестве значения вставьте имя пользователя реестра контейнеров Azure, которое было скопировано ранее. Выберите Добавить секрет.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-24-21-48-54.png)

1. Добавьте еще один секрет, введя имя ACR\_PASSWORD и вставив в качестве значения пароль реестра контейнеров Azure, который был скопирован ранее.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-24-21-51-24.png)

1. В сеансе Azure Cloud Shell, подключенном к виртуальной машине агента сборки, перейдите в каталог ~ / Fabmedical:

cd ~/Fabmedical

1. Перед настройкой рабочих процессов GitHub Actions необходимо создать каталог .github/workflows, если он еще не существует. Сделайте это, выполнив следующие команды:
2. mkdir ~/Fabmedical/.github

mkdir ~/Fabmedical/.github/workflows

1. Перейдите в каталог .github / workflows:

cd ~/Fabmedical/.github/workflows

1. Затем создайте YAML-файл рабочего процесса.
2. vi content-web.yml

Добавьте следующее в качестве содержимого. Обязательно замените следующие заполнители:

* + замените [SHORT\_SUFFIX] на ваш короткий суффикс, например SOL.

name: content-web

# This workflow is triggered on push to the 'content-web' directory of the main branch of the repository

on:

push:

branches:

- main

paths:

- 'content-web/\*\*'

# Configure workflow to also support triggering manually

workflow\_dispatch:

# Environment variables are defined so that they can be used throughout the job definitions.

env:

imageRepository: 'content-web'

resourceGroupName: 'fabmedical-[SHORT\_SUFFIX]'

containerRegistryName: 'fabmedical[SHORT\_SUFFIX]'

containerRegistry: 'fabmedical[SHORT\_SUFFIX].azurecr.io'

dockerfilePath: './content-web'

tag: '${{ github.run\_id }}'

# Jobs define the actions that take place when code is pushed to the main branch

jobs:

build-and-publish-docker-image:

name: Build and Push Docker Image

runs-on: ubuntu-latest

steps:

# Checkout the repo

- uses: actions/checkout@master

- name: Set up Docker Buildx

uses: docker/setup-buildx-action@v1

- name: Login to ACR

uses: docker/login-action@v1

with:

registry: ${{ env.containerRegistry }}

username: ${{ secrets.ACR\_USERNAME }}

password: ${{ secrets.ACR\_PASSWORD }}

- name: Build and push an image to container registry

uses: docker/build-push-action@v2

with:

context: ${{ env.dockerfilePath }}

file: "${{ env.dockerfilePath }}/Dockerfile"

pull: true

push: true

tags: |

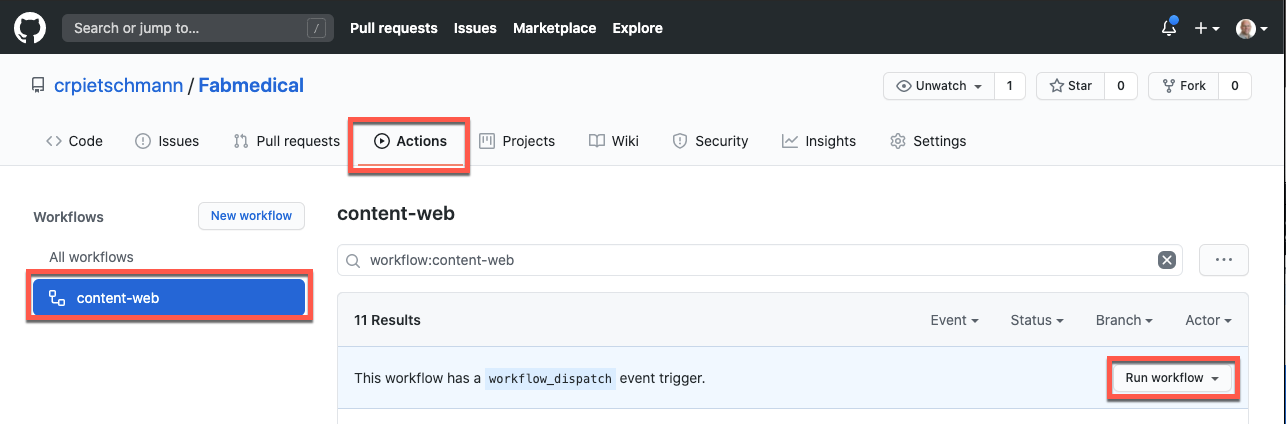
${{ env.containerRegistry }}/${{ env.imageRepository }}:${{ env.tag }}

${{ env.containerRegistry }}/${{ env.imageRepository }}:latest

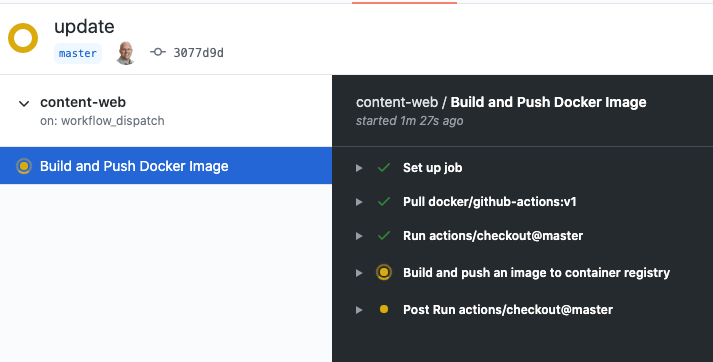
1. Сохраните файл и выйдите.
2. Сохраните конвейер YAML, затем выполните фиксацию и отправьте его в репозиторий Git:
3. git add .
4. git commit -m "Added workflow YAML"

git push

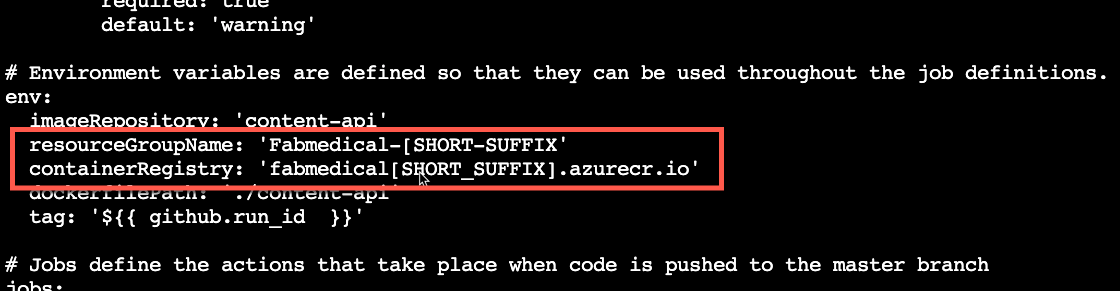
1. В GitHub вернитесь на экран репозитория и выберите вкладку Действия.
2. На странице "Действия" выберите рабочий процесс контент-веб.
3. В рабочем процессе контент-веб выберите Запустить рабочий процесс и вручную запустите рабочий процесс для выполнения.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-25-15-38-06.png)

1. Через секунду в списке отобразится только что запущенное выполнение рабочего процесса. Выберите новое исполнение Content-Web, чтобы просмотреть его статус.
2. При выборе задания Build and Push Docker Image рабочего процесса будет отображаться состояние его выполнения.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-25-15-42-11.png)

1. Затем настройте рабочий процесс content-api. Этот репозиторий уже включает content-api.yml, расположенный в каталоге .github/workflows. Откройте файл .github/workflows/content-api.yml для редактирования.
2. Измените значения среды resourceGroupName и containerRegistry, заменив [SHORT\_SUFFIX] вашим собственным трехбуквенным суффиксом, чтобы он соответствовал имени вашего реестра контейнеров и группе ресурсов.

[](https://github.com/microsoft/MCW-Cloud-native-applications/blob/main/Hands-on%20lab/media/2020-08-25-15-59-56.png)

1. Зафиксируйте и отправьте изменения в репозиторий Git:
2. git add .
3. git commit -m "Updated workflow YAML"

git push

1. Сохраните файл, затем перейдите к репозиториям в GitHub, выберите Действия, а затем вручную запустите рабочий процесс content-api.