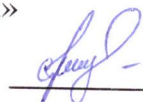


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект забезпечення безпеки підключаємих
пристроїв IoT до хмарного середовища Azure»

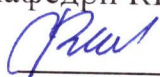
Виконав студент 4-го курсу, групи КБ-1
спеціальності 125 «Кібербезпека»

 А.І. Косачев

Керівник к.т.н., доцент, доцент кафедри КІ та КБ

 І.В. Пулеко

Рецензент старший викладач кафедри КІ та КБ

 В.О. Філіпов

Житомир – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 125 «КІБЕРБЕЗПЕКА»

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерної інженерії
та кібербезпеки

Єфіменко А. А. 
«12» квітня 2021 р.

ЗАВДАННЯ
на випускню кваліфікаційну роботу

Студента Косачева Антона Ігоровича _____

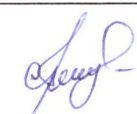
Тема роботи: Проект забезпечення безпеки підключаємих пристроїв IoT до хмарного середовища Azure _____

Затверджена Наказом університету від «12» квітня 2021 р. № 134/С

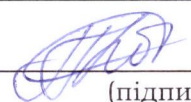
Термін здачі студентом закінченої роботи 25 червня 2021 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження) Хмарне середовище, симулятор Visual Studio Code, пристрої IoT.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Пулеко І.В доцент кафедри КІ та КБ	12.04.21 	12.04.21 
2	Пулеко І.В доцент кафедри КІ та КБ	12.04.21 	12.04.21 
3	Пулеко І.В доцент кафедри КІ та КБ	12.04.21 	12.04.21 

Керівник _____


(підпис)

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Розробка та постановка плану, щодо роботи. Пошук, опрацювання та детальний аналіз інформації, яка відноситься до нашої роботи	5 квітня 2021 – 17 квітня 2021	Виконано
2	Розробка методики безпечного підключення пристроїв	18 квітня 2021 – 29 квітня 2021	Виконано
3	Створення безпечного підключення пристроїв	31 квітня 2021 – 10 травня 2021	Виконано
4	Тестування	15 травня 2021 – 25 травня 2021	Виконано
5	Захист		

Студент _____

(підпис)

Керівник _____

(підпис)

РЕФЕРАТ

Випускна робота бакалавра складається з проекту забезпечення безпеки підключаємих пристроїв IoT до хмарного середовища Azure.

Пояснювальна записка має 8 сторінок додатків. Список використаних джерел містить 15 найменувань і займає 2 сторінки. В роботі наведено 22 рисунків. Загальний обсяг роботи – 81 сторінка.

Метою роботи є забезпечення безпеки підключаємих пристроїв IoT до хмарного середовища Azure.

Об'єктом дослідження є процес забезпечення безпеки підключення пристроїв IoT до хмарного середовища Azure.

Предметом дослідження є процес створення сертифікатів для безпечного підключення пристроїв IoT до хмарного середовища Azure.

В роботі визначено основні поняття, щодо самих пристроїв IoT, хмарного середовища Azure та проаналізовано методи вирішення поставленої задачі. Наведено теоретичні відомості про розробку методики безпечного підключення. Також описана програмна реалізація забезпечення безпеки підключаємих пристроїв.

Наводяться результати даної роботи.

КЛЮЧОВІ СЛОВА: ХМАРНЕ СЕРДОВИЩЕ AZURE, ПРИСТРОЇ IOT, X.509, DPS, КОРЕНЕВИЙ СЕРТИФІКАТ.

					КБ.КР.Б – 125 – 20 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дат	Проект забезпечення безпеки підключаємих пристроїв IoT до хмарного середовища Azure	Літ.	Арк.	Акрушів	
Розроб.		Косачев А.І		29.06.21					
Керівник		Пулєко І.В		29.06.21				69	
Рецензент		Філіпов В.О		29.06.21		Житомирська політехніка, група КБ - 1			
Зав. Каф.		Єфіменко А.А		29.06.21					

ABSTRACT

The Bachelor Robot release is based on a project to secure IoT add-ons in the bleak golden mean of Azure.

The note is explained by 8 pages of documents. List of winners dzherel for revenge 15 new loans and loans of 2 parties. There are 22 drawings in the robot. Robocalling of the robot - 81 sides.

I use robots to secure IoT add-ons in a dull Azure environment. About the process of securing the connection of IoT attachments to the Azure dark environment.

The next step is the process of obtaining certifications to securely connect IoT investments to the cold golden mean of Azure.

In the robot, the basic understanding is given, as well as the IoT investments themselves, the dark middle class of Azure, the methods for solving the tasks are analyzed. Theoretical ideas about the development of the unburned joint method are presented. It also describes software for implementing the security of connected attachments. Guided by the results of this robot.

KEY WORDS: HMARNE SERDOVISCHIE AZURE, PRISTROÏ IOT, X.509, DPS, ROOT CERTIFICATE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	10
ВСТУП.....	11
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ РОБОТИ	13
1.1. Обґрунтування актуальності теми роботи	13
1.2. Аналіз особливостей хмарного середовища Azure та засобів управління його безпекою	14
1.3. Аналіз особливостей стратегій і рішень Інтернету речей Azure IoT.....	17
1.4. Аналіз еталонної архітектури Microsoft Azure Internet of Things (IoT).....	19
1.5. Постановка завдання	23
Висновки по 1 розділу.....	23
РОЗДІЛ 2. РОЗРОБКА МЕТОДИКИ БЕЗПЕЧНОГО ПІДКЛЮЧЕННЯ ПРИСТРОЇВ IoT В AZURE	25
2.1. Аналіз особливостей сертифікації X.509	25
2.2. Розробка методики безпечного підключення пристроїв IoT з використанням Служби підготовки пристроїв DPS	32
2.3. Створення служби підготовки пристроїв в центрі Інтернету речей Azure та кореневого сертифікату	39
2.4. Створення декількох кінцевих сертифікатів	41
Висновки по 2 розділу.....	43
РОЗДІЛ 3. ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ ТА ЕФЕКТИВНОСТІ ПРИЙНЯТИХ РІШЕНЬ	57
3.1. Розробка коду для пристроїв датчиків	57
3.2. Тестування автоматичної підготовки та призначення в Центр Інтернету речей для декількох пристроїв	61

Висновки по 3 розділу.....	62
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	63
ДОДАТКИ.....	65

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення

ПК – персональний комп'ютер

ОС – операційна система

DPS – Device Provisioning Service

AD – Active directory

					КБ.КР.Б – 125 – 20 – ПЗ	Арк.
						10

ВСТУП

На сьогоднішній день дуже важко уявити роботу сучасних пристроїв в багатьох компаніях без використання технологій, пов'язаних з хмарним середовищем. Адже з кожним днем у світі, сотні тисяч компаній збільшуються у своїх масштабах, використовують все більше пристроїв, все більше ресурсів, все більше інформації, процеси стають все складнішими та витрати на це неймовірно зростають. Кожна людина, кожен власник бізнесу, кожен користувач зацікавлений у тому, щоб економно та розумно витрачати свої ресурси та одночасно отримувати доволі гарний результат.

У світі існує багато різних варіантів вирішення усіх цих моментів, але дійсно гарних виходів з цього положення дуже мало. Багато людей, з правильними помислами, оптимізувати свої витрати та запровадити нові технології у свою справу, але не дуже якісно розібравшись в питанні та обравши неправильний напрямок змін, попадає в ще гіршу ситуацію та його проблеми зростають ще більше.

Сучасний розвиток світових технологій дає нам безмежний вибір різноманітних платформ, технологій і т.п. Один з таких варіантів є Microsoft Azure. Саме це є темою нашої дипломної роботи, «Проект забезпечення безпеки підключаємих пристроїв IoT до хмарного середовища Azure».

Наша задача якомога детальніше та якісніше розібратись у цьому питанні. Визначити та вивчити усі теоретичні відомості, Максимально точно виявити усі переваги та недоліки, саме цього напрямку. Розробити та створити проект забезпечення пристроїв до хмарного середовища. Перевірити та протестувати всю зроблену нами роботу та зробити висновки.

Метою випускної кваліфікаційної роботи є підвищення рівня захищеності підключаємих пристроїв IoT до хмарного середовища Azure.

Для досягнення мети в роботі виконані такі **завдання**:

- проведено обґрунтування актуальності теми роботи;
- аналіз особливостей хмарного середовища Azure та засобів управління його безпекою;

- аналіз еталонної архітектури Microsoft Azure Internet of Things;
- розроблено методику безпечного підключення пристроїв IoT в Azure;
- перевірка працездатності та ефективності прийнятих рішень.

Об'єктом дослідження є процес забезпечення безпеки підключення пристроїв IoT до хмарного середовища Azure.

Предметом дослідження є процес створення сертифікатів для безпечного підключення пристроїв IoT до хмарного середовища Azure.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ РОБОТИ

1.1. Обґрунтування актуальності теми роботи

Тема цієї дипломної роботи розглядає одне з найактуальніших питань, щодо розробки та забезпечення безпеки підключаємих пристроїв. Взагалі, що таке пристрої IoT (Інтернет речей) та хмарне середовище Azure?

Пристрої IoT - це концепція мережі передачі даних між фізичними об'єктами, оснащеними вбудованими засобами і технологіями для взаємодії один з одним або з зовнішнім середовищем.

Хмарне середовище Azure - це платформа хмарних обчислень з постійно розширюємим набором служб, яка допоможе Вам створювати рішення для ваших бізнес-завдань.

На сьогоднішній день питання розвитку хмарних технологій одне з найбільш поширених в сфері ІТ. Під поняттям хмарні обчислення, хмара, хмарні технології та хмарні середовища мається на увазі надання людині (користувачу) певних ресурсів та потужностей у вигляді інтернет-сервісу. На мою думку актуальність хмарних технологій пов'язана не тільки з розвитком та масштабованістю інформаційних технологій, а також зі зниженням витрат на це. Хмарне середовище є дуже зручним інструментом для компаній, які вирішили перебудуватись та оптимізувати свої ресурси шляхом перенесення обчислювальних потужностей в хмарний дата-центр.

Перехід від стандартної локальної інфраструктури до хмарного середовища значно зменшує капітальні витрати не тільки на обладнання, а також на експлуатаційні витрати зв'язані з технічним обслуговуванням. Та дає змогу використовувати необхідні ресурси по мірі необхідності.

Саме ці переваги Microsoft Azure надає своїм користувачам.

1.2. Аналіз особливостей хмарного середовища Azure та засобів управління його безпекою

Azure - це платформа хмарних обчислень з постійно розширюється набором служб, яка допомагає користувачам створювати рішення для їх бізнес-завдань. Служби Azure можуть бути простими веб-службами для розміщення нашого бізнесу в хмарі або повністю віртуалізованими комп'ютерами для виконання призначених для користувача програмних рішень. Azure надає безліч хмарних служб, таких як:

- віддалене сховище;
- розміщення баз даних;
- централізоване управління обліковими записами.

Azure також пропонує нові можливості, такі як ІІІ та Інтернет речей.

Хмарні обчислення - це надання обчислювальних служб через Інтернет за допомогою цінової моделі з оплатою в міру використання. Зазвичай ми платимо тільки за використовувані хмарні служби. Такий підхід дозволяє нам:

- скоротити експлуатаційні витрати;
- ефективніше обслуговувати інфраструктуру;
- змінювати масштаб в залежності від потреби бізнесу.

Іншими словами, хмарні обчислення - це спосіб орендувати обчислювальні ресурси та сховище в сторонньому центрі обробки даних. Завершивши їх використання, ми повертаємо їх назад. Рахунки виставляються тільки в міру використання.

Замість того щоб використовувати фізичні ЦП та сховища в своєму центрі обробки даних, ми орендуємо їх на потрібний нам час. Всі турботи, пов'язані з підтриманням нашої інфраструктури, бере на себе постачальник хмари. Хмара дозволяє швидко вирішувати складні бізнес-завдання і пропонує користувачам найсучасніші рішення. При використанні Azure ми отримуємо все необхідне для створення чергового чудового рішення.

Azure надає більше 100 служб, що дозволяють вирішувати будь-які завдання: від запуску існуючих додатків в віртуальних машинах до освоєння нових парадигм програмного забезпечення, таких як інтелектуальні боти і змішана реальність.

Багато команд приступають до вивчення хмари, переміщаючи свої наявні програми на віртуальні машини, які працюють в Azure. Незважаючи на те, що перенесення наявних додатків в віртуальні машини - це оптимальний варіант, хмара є не просто альтернативним середовищем для роботи віртуальних машин.

Наприклад, платформа Azure надає служби ІІ та машинного навчання, що дозволяють вести природню взаємодію з користувачами за допомогою:

- зору;
- слуху;
- мовлення.

Вона також надає рішення для зберігання, які динамічно збільшуються для розміщення великих обсягів даних. Служби Azure реалізують рішення, які неможливі без хмари.

Портал Azure - це єдина веб-консоль, яку можна використовувати замість засобів командного рядка. На порталі Azure можна управляти підпискою Azure за допомогою графічного для користувача інтерфейсу. На порталі можна зробити наступне:

- створювати, адмініструвати та відстежувати всі рішення, починаючи від простих веб-додатків та закінчуючи складними хмарними розгортання;
- створювати настроювані панелі моніторингу для упорядкованого уявлення ресурсів;
- налаштовувати більш доступний режим для оптимальної роботи.

Портал Azure призначений для забезпечення стійкості і постійної доступності. Він забезпечує присутність в кожному центрі обробки даних Azure. Завдяки такій конфігурації портал Azure стійкий до збоїв окремих центрів обробки даних та дозволяє уникнути затримок в мережі через близьке

розташування до користувачів. Портал Azure постійно оновлюється та не відключається для виконання обслуговування.

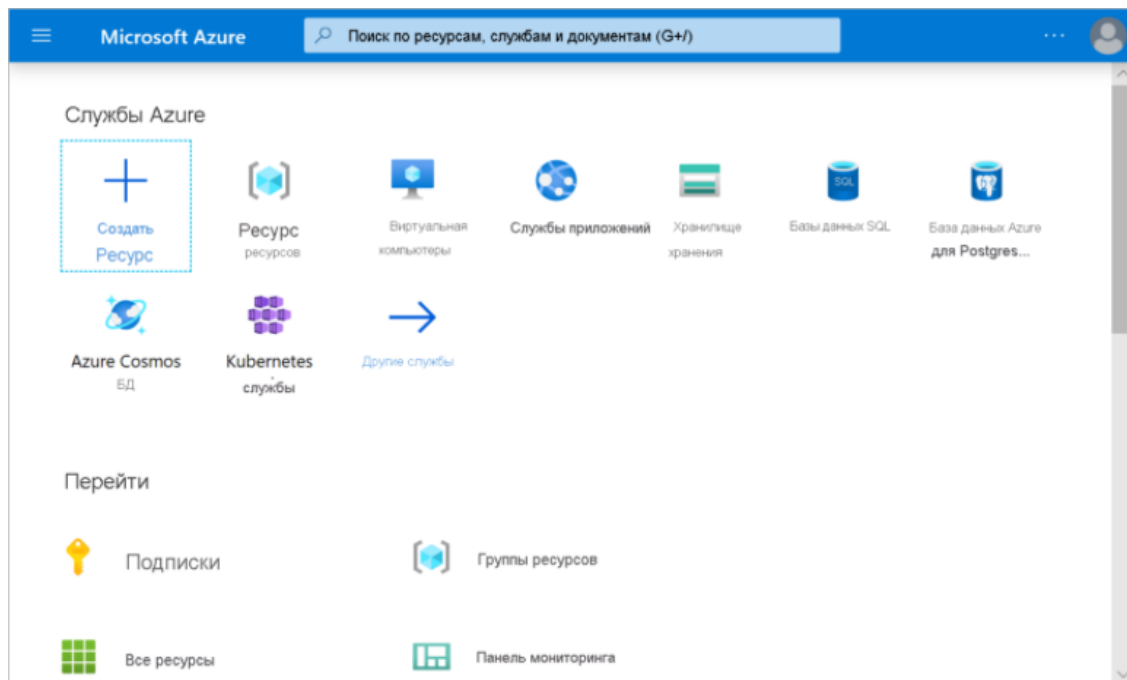


Рисунок 1.1 – Службы Azure

Портал Azure призначений для забезпечення стійкості і постійної доступності. Він забезпечує присутність в кожному центрі обробки даних Azure. Завдяки такій конфігурації портал Azure стійкий до збоїв окремих центрів обробки даних і дозволяє уникнути затримок в мережі через близьке розташування до користувачів. Портал Azure постійно оновлюється і не відключається для виконання обслуговування.

Існують найбільш популярні служби Azure:

- обчислення;
- мережа;
- сховище;
- мобільні додатки;
- бази даних;
- веб-служби;
- інтернет речей;
- великі дані;
- II;
- DevOps.

Обчислення. Служби обчислень - це одна з основних причин того, що компанії переходять на платформу Azure. Azure надає широкий вибір варіантів для розміщення додатків та служб.

Мережа. Зв'язування обчислювальних ресурсів та надання доступу до додатків - це ключова функція мереж Azure. Функції мереж в Azure включають в себе ряд варіантів для з'єднання зовнішнього світу зі службами та компонентами в глобальних центрах обробки даних Azure.

Сховище. Azure надає чотири основних типи служб сховища:

- сховище BLOB-об'єктів Azure;
- сховище файлів Azure;
- сховище черг Azure;
- табличне сховище Azure.

Мобільні додатки. Завдяки Azure розробники можуть швидко та легко створювати мобільні серверні служби для додатків iOS, Android та Windows. Функції, які раніше забирали багато часу та підвищували ризики для проекту, наприклад додавання корпоративного єдиного входу та підключення до локальних ресурсів, таким як SAP, Oracle, SQL Server і SharePoint, тепер легко використовувати.

Бази даних. Azure надає кілька служб баз даних для зберігання різноманітних типів даних та томів. А з можливостями глобального з'єднання ці дані доступні користувачам миттєво.

Веб-служби. У сучасному світі для бізнесу критично важливо бути гідно представленими в Інтернеті. Azure пропонує першокласну підтримку для створення та розміщення веб-додатків та веб-служб на базі HTTP.

Інтернет речей. Збільшення обсягу інформації, доступного користувачам. Персональні цифрові помічники перетворилися в смартфони, а тепер нам доступні розумний годинник, розумний термостат та навіть розумний холодильник. Повсюдне використання персональних комп'ютерів. Можливість доступу до цінної інформації будь-яких пристроїв, що підтримують

підключення до Інтернету. Концепція Інтернету речей полягає в можливості збору даних пристроями та їх передачі для аналізу.

Великі дані. Дані можуть мати будь-який формат та розмір. Під великими даними ми розуміємо великі обсяги даних. В метеорологічних та комунікаційних системах, геномних дослідженнях, на платформах візуалізації та в багатьох інших сценаріях можуть генеруватися сотні гігабайтів даних. При таких обсягах значно ускладнюються процеси аналізу даних та прийняття рішень. Найчастіше обсяг даних настільки великий, що традиційні підходи до обробки та аналізу просто не працюють.

II. Можливості штучного інтелекту в контексті хмарних обчислень реалізуються на базі широкого спектра служб, в основі яких лежать технології машинного навчання. Машинне навчання - це спосіб обробки та аналізу даних, який дозволяє комп'ютерам використовувати наявні дані для прогнозування майбутньої поведінки, результатів та трендів. Використовуючи машинне навчання, комп'ютери вчаться, не будучи явно запрограмованими.

Прогнози машинного навчання дозволяють зробити програми та пристрої ефективніше. Наприклад, при покупках через Інтернет машинне навчання допомагає рекомендувати інші продукти, які можуть вам сподобатися, на основі вже придбаних вами товарів. Або при крадіжці кредитної картки машинне навчання порівнює транзакцію з базою даних транзакцій та допомагає виявляти шахрайство. Коли робот-пилосос прибирає кімнату, машинне навчання дозволяє визначити, коли цей процес закінчився.

DevOps. Концепція DevOps дозволяє об'єднати людей, процеси, технології та можливості автоматизації доставки ПО для забезпечення безперервної підтримки користувачів. За допомогою Azure DevOps ви можете створювати конвеєри збірки і випуску, що забезпечують безперервну інтеграцію, доставку і розгортання ваших додатків. Ви можете інтегрувати тести репозиторіїв і додатків, здійснювати моніторинг програм і працювати з артефактами збірки.

Крім того, ми можемо працювати з елементами невиконаної роботи з метою відстеження, автоматизувати розгортання інфраструктури та здійснювати інтеграцію з широким спектром сторонніх коштів та служб, таких як Jenkins і Chef. Всі ці та багато інших функцій тісно інтегруються з платформою Azure, забезпечуючи узгоджене регулярне розгортання додатків та тим самим, оптимізацію процесів складання та випуску.

ІоТ дозволяє пристроям збирати та посилати відомості для аналізу даних. Інтелектуальні пристрої оснащені датчиками, які збирають дані. Нижче перераховані деякі поширені датчики, які вимірюють фізичні параметри:

- датчики навколишнього середовища, які реєструють рівні температури та вологості;
- сканери штрих-кодів, QR-кодів або оптичного розпізнавання символів (OCR);
- датчики географічного розташування та близькості;
- датчики освітлення, кольору та інфрачервоні датчики;
- звукові та ультразвукові датчики;
- датчики руху і торкання;
- акселерометри та датчики нахилу;
- датчики диму, газу та концентрації алкоголю;
- датчики помилок для виявлення проблем з пристроєм;
- механічні датчики для виявлення відхилень або деформацій;
- датчики витрати, рівня та тиску для вимірювання параметрів газів та рідин.

Припустимо, що наша компанія виробляє та обслуговує "розумні" торгові автомати з холодильниками. Яку інформацію нам необхідно відстежувати? Наприклад, ми хочемо переконатися в наступному:

- кожен торговий автомат працює без помилок;
- торгові автомати були скомпрометовані;
- системи охолодження торговельних автоматів підтримують певний діапазон температури для збереження вмісту;

- ми отримуємо повідомлення, коли наявність товарів досягає певного рівня, щоб поповнити запаси.

Так як торговий автомат може збирати та відправляти ці відомості у вигляді стандартного повідомлення, такі повідомлення можна отримувати, зберігати, упорядковувати та відображати за допомогою служб Інтернету речей Azure.

Дані, що збираються з цих пристроїв, можна об'єднати зі службами ІІІ Azure, щоб допомогти нам прогнозувати наступне:

- коли торговим автоматам потрібно профілактичне обслуговування;
- коли потрібно поповнити запаси і замовити новий товар у постачальників.

У Azure існує багато служб, які допоможуть нам реалізувати комплексні рішення для Інтернету речей.

Центр Інтернету речей - це розміщена в хмарі керована служба, яка діє в якості центру обробки повідомлень для двостороннього зв'язку між додатком Інтернету речей та пристроями, якими воно управляє. Центр Інтернету речей Azure можна використовувати для створення рішень Інтернету речей з надійним та безпечним зв'язком між мільйонами пристроїв Інтернету речей та серверною частиною хмарних рішень. До Центру Інтернету речей можна підключити практично будь-який пристрій.

Служба Центру Інтернету речей підтримує передачу даних з пристрою в хмару та з хмари на пристрій. Крім того, вона підтримує кілька шаблонів повідомлень, таких як відправка даних телеметрії з пристрою в хмару та відправлення файлів з пристроїв, а також методи запиту та відповіді для управління пристроями з хмари. Коли Центр Інтернету речей отримує повідомлення від пристрою, він може перенаправляти їх в інші служби Azure.

З точки зору передачі даних із хмари на пристрій Центр Інтернету речей забезпечує управління та контроль. Тобто ви можете вручну або автоматично дистанційно керувати підключеними пристроями, щоб відкривати клапани, задавати температуру, перезапускати завислий пристрій і т.д.

Моніторинг Центру Інтернету речей дозволяє підтримувати працездатність рішення шляхом відстеження подій, таких як створення пристрою, збої в роботі пристроїв та підключення пристроїв.

Azure IoT Central побудований на основі Центру Інтернету речей, додаючи панель моніторингу, яка дозволяє підключатися до пристроїв Інтернету речей, відстежувати їх та управляти ними. Візуальний інтерфейс дозволяє швидко підключати нові пристрої та стежити за переданими ними даними телеметрії або повідомленнями про помилки. Ми можемо спостерігати за загальною продуктивністю всіх пристроїв в цілому та налаштовувати оповіщення, які відправляються, коли той або інший пристрій вимагає обслуговування. Нарешті, можна відправляти оновлення вбудованого ПЗ на пристрій.

Щоб допомогти нам швидко приступити до роботи, IoT Central надає початкові шаблони для поширених сценаріїв в різних галузях, таких як роздрібна торгівля, енергетика, охорона здоров'я та державне управління. Ці початкові шаблони можна налаштувати безпосередньо в інтерфейсі, вибравши одну з існуючих тем або створивши власну призначену для користувача тему, надавши логотип і т.д. IoT Central дозволяє адаптувати початкові шаблони для конкретних даних, що відправляються з пристроїв, потрібних звітів та повідомлень.

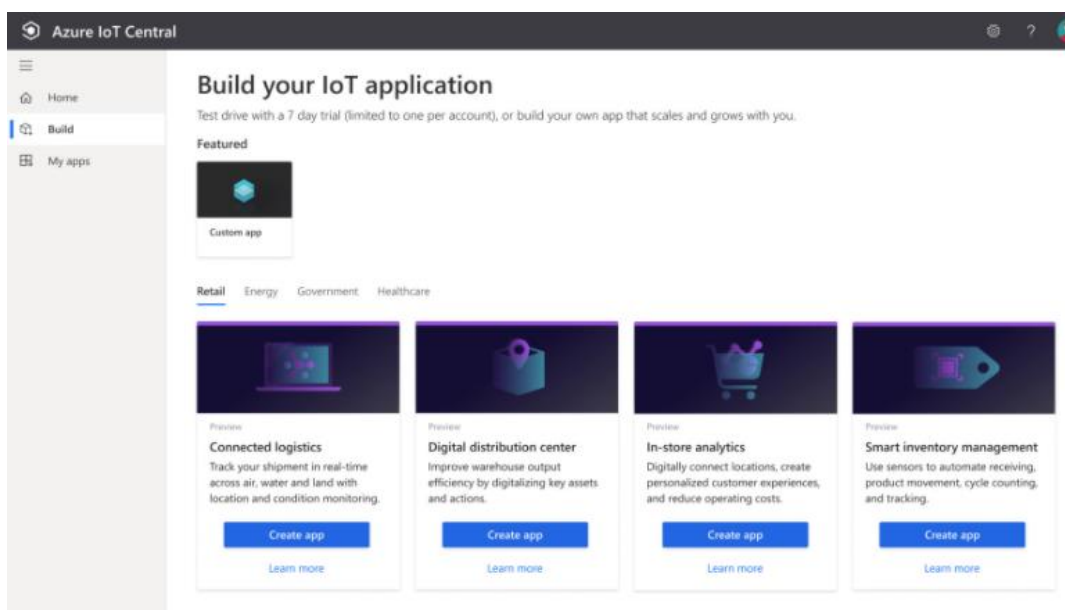


Рисунок 1.2 – Azure IoT Central

Ми можемо управляти пристроями віддалено за допомогою призначеного для користувача інтерфейсу. Ця функція дозволяє відправити оновлення програмного забезпечення або змінити властивість пристрою. Ми можемо змінити необхідну температуру для одного або всіх торгових автоматів з системою охолодження безпосередньо з IoT Central.

Ключовим елементом при роботі з IoT Central є використання шаблонів пристроїв. Шаблони пристроїв дозволяють підключати пристрої без написання коду на стороні служби. У IoT Central шаблони застосовуються для створення панелей моніторингу, оповіщення та т. Д. Проте розробникам пристроїв потрібно написати код, який буде виконуватися на пристроях, причому він повинен відповідати специфікації шаблону пристрою.

Azure Sphere утворює комплексне рішення Інтернету речей з високим рівнем безпеки для клієнтів, яке охоплює всі аспекти: від обладнання та операційної системи пристрою до безпечного методу передачі повідомлень з пристрою в центр повідомлень. Azure Sphere має вбудовані функції взаємодії та захисту для пристроїв, підключених до Інтернету. Azure Sphere складається з трьох частин:

1. Перша частина - це мікроконтролер (MCU) Azure Sphere, який відповідає за роботу операційної системи і обробку сигналів від підключених датчиків. Нижче показаний мікроконтролер Seeed Azure Sphere MT3620 з комплекту SDK. Це один з декількох початкових наборів, доступних для створення прототипів і розробки додатків Azure Sphere.



Рисунок 1.3 - Мікроконтролер (MCU) Azure Sphere.

2. Друга частина - це настраюється операційна система (ОС) Linux, яка відповідає за взаємодію зі службою безпеки і виконання програмного забезпечення постачальника.

3. Третя частина - це служба безпеки Azure Sphere, також звана AS3. Її завдання полягає в запобіганні компрометації пристрої зловмисником. Коли пристрій намагається встановити зв'язок із Azure, воно спочатку має пройти перевірку справжності на основі сертифікату. Якщо перевірка справжності виконана успішно, AS3 перевіряє, чи не було пристрій незаконно змінено. Після встановлення безпечного каналу зв'язку AS3 передає на пристрій поновлення для ОС або схваленого клієнтського програмного забезпечення.

Після того, як система Azure Sphere підтвердить автентичність пристрою, пристрій зможе взаємодіяти з іншими службами Інтернету речей Azure, відправляючи дані телеметрії та відомості про помилки.

Центр безпеки Azure - це служба моніторингу, що дозволяє спостерігати за станом безпеки у всіх ваших службах, як в Azure, так і в локальному середовищі. Термін "стан безпеки" відноситься до елементів управління і політикам кібербезпеки, а також до того, наскільки добре ми можемо прогнозувати загрози безпеки, запобігати їм та реагувати на них. Існують такі можливості центру безпеки:

- моніторинг параметрів безпеки для локальних та хмарних робочих навантажень;
- автоматичне застосування необхідних налаштувань безпеки до нових ресурсів у міру їх появи в мережі;
- надання рекомендацій щодо забезпечення безпеки на основі поточних конфігурацій, ресурсів та мереж;
- безперервне відстеження наших ресурсів та виконання автоматичних оцінок безпеки для визначення потенційних вразливостей до того, як ними зможуть скористатися зловмисники;
- використання машинного навчання для виявлення та блокування установки шкідливих програм на віртуальних машинах та інших ресурсах.

Можна за допомогою адаптивних елементів управління додатками визначати правила, в яких перераховані дозволені додатки, щоб могли запускатися тільки додатки, які ви дозволили;

- виявлення та аналіз потенційних входять атак, а також дослідження загроз та будь-яких дій після порушень, які можуть статися;

- забезпечення оперативного управління доступом до мережевих портів.

Таким чином, мережа буде вирішувати тільки необхідний трафік і тоді, коли він необхідний, що дозволяє скоротити кількість напрямків атак.

Tailwind Traders може використовувати Центр безпеки, щоб отримувати детальний аналіз різних компонентів в своєму середовищі. Оскільки ресурси компанії аналізуються з точки зору засобів управління безпекою будь-яких призначених нею політик управління, співробітники компанії можуть спостерігати за загальним відповідністю нормативним вимогам з точки зору безпеки з одного місця.

Политика и соответствие требованиям

Общая оценка безопасности



[Изучите результаты оценки безопасности >](#)

Соответствие нормативным требованиям

PCI DSS 3.2.1	34 из 45: контроль пройден
Azure CIS 1.1.0	20 из 24: контроль пройден
SOC TSP	12 из 13: контроль пройден

Покрытие подписок



Рисунок 1.4 – Центр безпеки Azure.

Припустимо, що компанія Tailwind Traders повинна дотримуватися стандарт захисту безпеки в індустрії платіжних карт (PCI DSS). Цей звіт показує, що у компанії є ресурси, які необхідно виправити.

У розділі Гігієна безпеки ресурсів Tailwind Traders може побачити стан своїх ресурсів з точки зору безпеки. Рекомендації розділені на категорії важливості "низька", "середня" та "висока", щоб спростити призначення пріоритетів дій щодо виправлення. Приклад наведено нижче.



Рисунок 1.5 – Гігієна безпеки ресурсів.

Оцінка безпеки - це показник стану безпеки організації. Оцінка безпеки виконується за допомогою елементів управління безпекою або груп пов'язаних рекомендацій з безпеки. Наша оцінка - це процентна частка елементів управління безпекою, яким ви відповідаєте. Чим більшій кількості елементів управління безпекою ви відповідаєте, тим вище буде ваша оцінка. Оцінка покращується після внесення виправлень у відповідності з усіма рекомендаціями для одного ресурсу в елементі управління. Ось приклад з порталу Azure, де показана оцінка 57 відсотків, або 34 бали з 60.

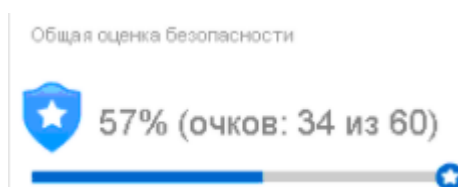


Рисунок 1.6 – Загальна оцінка безпеки.

Дотримання рекомендацій оцінки безпеки допомагає захистити нашу організацію від загроз. За допомогою централізованої панелі моніторингу в Центрі безпеки Azure організації можуть спостерігати за безпекою своїх ресурсів Azure, таких як посвідчення, дані, додатки, пристрої та інфраструктура, а також працювати над нею з метою поліпшення. Оцінка безпеки допомагає:

- дізнаватися про поточний стан безпеки вашої організації;
- поліпшувати стан безпеки, надаючи можливості виявлення, моніторингу та контролю, а також відповідні рекомендації;
- виконувати порівняння з еталонами та встановлювати ключові показники ефективності (КПЕ).

Центр безпеки включає розширені можливості хмарної захисту для віртуальних машин, мережевої безпеки та цілісності файлів. Tailwind Traders може використовувати Центр безпеки для централізованого перегляду всіх попереджень системи безпеки. Далі компанія може відхиляти хибні попередження, продовжувати їх розслідування, виправляти попередження вручну або використовувати автоматичний відповідь за допомогою автоматизації робочого процесу.

Автоматизація робочого процесу використовує Azure Logic Apps та з'єднувачі Центру безпеки. Додаток логіки може ініціюватися попередженням про виявлення загрози або рекомендацією Центру безпеки, відфільтрованої по імені або серйозності. Потім ви можете налаштувати це додаток логіки для виконання такої дії, як відправка електронного листа або публікація повідомлення в каналі Microsoft Teams.

1.3 Аналіз особливостей стратегій і рішень Інтернету речей Azure IoT

Інтернет речей (IoT) - це мережа пристроїв, підключених до Інтернету, які передають дані від датчиків в хмару для їх централізованої обробки. Зазвичай ці датчики вбудовані в пристрої моніторингу. Такі пристрої, які називаються пристроями IoT, володіють деякою обчислювальною потужністю для управління взаємодією. Генерований ними потік даних називається телеметрією. Найбільш часто використовуваними значеннями телеметрії є прискорення, вологість, розташування, тиск, температура і швидкість. Існує також безліч інших корисних показників.

З протилежного боку, в хмарі, є ресурс, званий Центром Інтернету речей, який збирає дані і обробляє їх. Центр також може обмінюватися даними з пристроями, наприклад для зміни параметрів або налаштувань.

Завдяки Інтернету речей один оператор за допомогою хмарного порталу може відстежувати та контролювати величезні масиви віддалених пристроїв. Інтернет речей дозволяє компаніям перевести свої операції на якісно новий рівень функціонування. Після збору даних їх можна проаналізувати для отримання інформативних аналітичних відомостей.

Ці аналітичні відомості можуть сприяти підвищенню ефективності, скорочення відходів або зниження витрат. Крім того, такі аналітичні відомості можуть допомогти в підвищенні безпеки інтелектуальних просторів. Будь то виробництво, транспортування або роздрібна торгівля, Інтернет речей Azure допомагає реалізувати нові бізнес-операції за допомогою рішень Інтернету речей, розроблених спеціально для потреб цієї галузі.

Інтернет речей Azure від Майкрософт допомагає використовувати наявні у вас навички, технології та ресурси для створення рішення Інтернету речей. Таке рішення може з легкістю отримувати дані від підключених пристроїв, а потім формувати на основі їх цінні аналітичні відомості для вашого бізнесу.

За допомогою Інтернету речей Azure можна створювати засоби аналітики для організації, використовуючи в рішенні машинне навчання і штучний

інтелект (ШІ). Ці сучасні технології дозволяють прискорити прогнозування, підвищити ефективність роботи, а також створити безпечну і надійну платформу. Центр Інтернету речей Azure може допомогти організаціям з будь-якої галузі у вирішенні широкого спектру бізнес-завдань, а також сприяє інноваціям і цифровому перетворенню.

Аналітичні відомості виходять в результаті обробки даних телеметрії на серверах. Ці аналітичні відомості можна перетворити в ефективні процеси та служби, які допомагають підвищити продуктивність і ефективність бізнесу. Інновації за допомогою Інтернету речей можна реалізувати в багатьох галузях. приклади галузей:

- виробництво;
- транспорт і логістика;
- енергетика та природні ресурси;
- сільське господарство;
- роздрібна торгівля;
- державний сектор;
- охорона здоров'я;
- фінансові послуги;
- розумні будинки і розумні міста.

Рішення Інтернету речей Azure, як правило, є горизонтальними та можуть охоплювати кілька галузей. Галузеві експерти можуть по-різному називати додатки Інтернету речей, проте зазвичай вони посилаються на п'ять поширених сценаріїв, перерахованих нижче. Далі в цьому модулі ми розглянемо застосування цих рішень за допомогою служб Інтернету речей Azure.

Завдяки профілактичному обслуговуванню та аналітиці потоків даних за допомогою Інтернету речей ви задаєте умови, при виконанні яких активуються певні дії. Ці дії, за умови їх оперативного виконання, перешкоджають збою обладнання. Крім того, ви можете перетворити аналітичні відомості, отримані на основі поточкових даних, в інформацію для виявлення змін режимів

експлуатації. У цьому випадку рішення застосовується не для запобігання збоїв, а для підвищення продуктивності.

За рахунок більш глибокого розуміння планування технічного обслуговування авіадвигунів клієнтам Rolls-Royce вдалося скоротити затримки рейсів і щорічного економити мільйони доларів.

Завдяки рішенням Інтернету речей Azure, включаючи Azure IoT Edge і Центр Інтернету речей Azure, міністерству транспорту штату Аляска вдалося знайти масштабується, Інтернету речей, яке дозволяє відстежувати потужні циклони.

Центр Інтернету речей Azure - це не тільки революційний засіб для підвищення результатів бізнесу, а й каталізатор для суттєвих перетворень в організації. Підключивши обладнання, верстати та інструменти, ви отримуєте аналітичні дані для впровадження інновацій і реалізації можливостей розвитку.

1.4 Аналіз еталонної архітектури Microsoft Azure Internet of Things (IoT)

На схемі нижче показана рекомендована архітектура рішення Інтернету речей, яка підійде багатьом підприємствам:



Рисунок 1.7 – Еталонна архітектура Microsoft Azure Internet of Things.

У цій еталонній архітектурі показана рекомендована архітектура для додатків Інтернету речей в Azure, яка використовує компоненти платформи як послуги (PaaS).

Додатки Інтернету речей можна описати як речі (пристрої) для передачі даних, які створюють аналітичні відомості. Ці відомості створюють дії для поліпшення компанії або процесів. Наприклад, модуль (річ), що відправляє дані температури. Ці дані використовуються для оцінки правильності роботи модуля (аналіз). Аналітичні відомості використовуються, щоб упереджаючи призначити пріоритет розкладу обслуговування для модуля (дія).

Ця еталонна архітектура використовує компоненти Azure PaaS (платформа як послуга). Ще один рекомендований варіант створення рішень IoT в Azure:

1. OT Central Azure. IoT Central - це повністю керована рішення SaaS (програмне забезпечення як послуга). Воно відволікає технічні рішення і дозволяє зосередитися виключно на власному. Однак разом з простотою ви отримуєте менше налаштовуються рішень, ніж в рішенні на основі PaaS.

На вищому рівні існує два способи обробки даних телеметрії: "гарячий" і "холодний" шлях. Різниця пов'язана з вимогами до затримки і доступу до даних.

2. "Гарячий" шлях дозволяє аналізувати дані в режимі реального часу в міру їх надходження. При застосуванні цього способу дані телеметрії повинні оброблятися з дуже низькою затримкою. "Гарячий" шлях зазвичай реалізується за допомогою підсистеми потокової обробки. Вихідні дані можуть ініціювати сповіщення або записуватися в структурований формат, який можна запросити за допомогою засобів аналітики.

3. "Холодний" шлях дозволяє виконати пакетну обробку протягом більш тривалих інтервалів (щогодини або щодня). "Холодний" шлях зазвичай працює з великими обсягами даних, проте результати не потрібні також своєчасно, як в "гарячому" шляху. В "холодному" шляху збираються необроблені дані телеметрії, а потім передаються в процес пакетної обробки.

Архітектура складається з наступних компонентів. Для деяких додатків можуть вимагатися лише деякі з перерахованих тут компонентів.

Пристрої ІОТ. Пристрої можна безпечно зареєструвати в хмарі і підключати до нього для відправки та отримання даних. Деякі пристрої можуть бути прикордонними пристроями, які виконують певну обробку даних на самому пристрої або в польовому шлюзі. Ми рекомендуємо використовувати Azure IoT Edge для обробки прикордонних пристроїв.

Хмарний шлюз. Хмарний шлюз надає центр хмарних обчислень пристроїв для безпечного підключення до хмари і відправки даних. Він також надає можливості управління пристроями, включаючи команди і контроль пристроїв. Для хмарного шлюзу ми рекомендуємо використовувати Центр Інтернету речей. Центр Інтернету речей - це розміщена хмарна служба, яка приймає дані про події з пристроїв, діючи як брокер повідомлень між пристроями і серверними службами. Центр Інтернету речей надає безпечне підключення, отримання даних подій, двосторонній обмін повідомленнями і управління пристроями.

Підготовка пристрою. При реєстрації та підключенні великого набору пристроїв ми рекомендуємо використовувати Службу підготовки пристроїв до додавання в Центр Інтернету речей (DPS). DPS дозволяє призначати і реєструвати пристрої в певні кінцеві точки Центру Інтернету речей Azure в потрібному масштабі.

Передача потокового обробка. Передача потокового аналізує великі потоки записів даних і оцінює правила для цих потоків. Для обробки потоків даних рекомендуємо використовувати Azure Stream Analytics. Stream Analytics може виконувати складний аналіз в необхідному масштабі за допомогою функцій обмеження вікна, агрегацій потоку і зовнішніх з'єднувачів джерел даних. Крім того, ви можете використовувати Apache Spark в Azure Databricks.

Машинне навчання дозволяє виконувати прогнозні алгоритми по відношенню до історичних даних телеметрії, включаючи такі сценарії, як

прогнозне обслуговування. Для роботи з машинним навчанням ми рекомендуємо використовувати службу Машинне навчання Azure.

Сховище "гарячого" шляху містить дані, які повинні бути відразу ж доступні на пристрої для створення звітів і візуалізації. Як сховище "гарячого" шляху ми рекомендуємо використовувати Cosmos DB. Cosmos DB - це багатомодельна глобально розподілена база даних.

Сховище "холодного" шляху містить дані, які зберігаються довгостроково і використовуються в пакетній обробці. Як сховище "холодного" шляху ми рекомендуємо використовувати сховище BLOB-об'єктів Azure. Дані можна заархівувати в сховище BLOB-об'єктів на невизначений час без зайвих витрат. Такі дані легко доступні для пакетної обробки.

При перетворенні даних виконуються дії з потоком даних телеметрії або його об'єднання. Приклади включають перетворення протоколів, наприклад, перетворення двійкових даних в JSON або об'єднання точок даних. Якщо дані необхідно перетворити до надходження в Центр Інтернету речей, ми рекомендуємо використовувати шлюз протоколу (не показано). В іншому випадку дані можна перетворити після надходження до Центру Інтернету речей. В цьому випадку ми рекомендуємо використовувати Функції Azure. Це рішення підтримує інтеграцію з Центром Інтернету речей, Cosmos DB і сховищем BLOB-об'єктів.

При інтеграції бізнес-процесів дії виконуються на основі аналітичних відомостей з даних пристрою. Сюди входить зберігання інформаційних повідомлень, створення оповіщень, відправка електронної пошти або текстових повідомлень або інтеграція з CRM. Для інтеграції бізнес-процесів ми рекомендуємо використовувати Azure Logic Apps.

Управління користувачами дозволяє настроїти обмеження користувачів або груп для виконання дій на пристроях, таких як оновлення вбудованого ПЗ. Воно також визначає можливості для користувачів в додатках. Для аутентифікації і авторизації користувачів ми рекомендуємо використовувати Azure Active Directory.

Засіб моніторингу безпеки Azure Security Center для Інтернету речей надає комплексне рішення по забезпеченню безпеки для робочих навантажень ІОТ і спрощує їх захист за рахунок уніфікованої видимості і контролю, адаптивної захисту від загроз, а також інтелектуального виявлення загроз і реагування між робочими навантаженнями з кінцевих пристроїв через прикордонні і через хмари.

1.5 Постановка завдання

У першому розділі, було проведено детальний аналіз предметної області теми та зроблений повний опис 4-х аспектів, таких як:

- обґрунтування актуальності теми;
 - аналіз особливостей хмарного середовища Azure та засобів управління його безпекою;
 - аналіз особливостей стратегій та рішень Інтернету речей Azure IoT;
 - аналіз еталонної архітектури Microsoft Azure Internet of Things (Інтернет речей).
- Були знайдені та наведені приклади у вигляді знімків екрану, схем і т.д.

Висновок до 1 розділу

Отже, провівши повний та детальний аналіз предметної області та обґрунтування актуальності теми, можна зробити певні висновки, щодо першого розділу.

По-перше, актуальність цього питання, на сьогоднішній день має високий рівень. С кожним днем по всій планеті мільйони людей та сотні тисяч великих, і не тільки великих, компаній впроваджують у свою справу новітні технології хмарних середовищ. Адже це значно спрощує працю людей. На мою думку саме ці впровадження є дуже зручним інструментом для користувачів, які вирішили змінити підхід до своєї справи, оптимізувати та більш розумно

використовувати свої ресурси шляхом перенесення обчислювальних потужностей в хмарний дата-центр.

По-друге, ми зробили детальний та повний опис теоретичних відомостей, щодо самого хмарного середовища. І можемо виявити найголовніші переваги Microsoft Azure.

Хмарне середовище Microsoft Azure - хмарна платформа та інфраструктура корпорації Microsoft, призначена для розробників застосунків хмарних обчислень і покликана спростити процес створення онлайнових додатків. Користувачі, які вирішили перейти від стандартної локальної інфраструктури до хмарного середовища значно зменшують свої витрати. Починаючи від витрат на обладнання та закінчуючи експлуатаційними витратами зв'язаними з технічним обслуговуванням цих технологій.

Існує ще одна перевага, а саме «Pay only for what you use». Що в перекладі на українську мову - «Плати за те, що використовуєш». Саме за таким принципом працює Microsoft Azure, що дає змогу своїм користувачам відстежувати та контролювати свою витрати, що в майбутньому економить великі кошти.

РОЗДІЛ 2. РОЗРОБКА МЕТОДИКИ БЕЗПЕЧНОГО ПІДКЛЮЧЕННЯ ПРИСТРОЇВ IoT В AZURE

2.1. Аналіз особливостей сертифікації X.509

X.509 - це стандартний формат для сертифікатів відкритих ключів, які надійно пов'язують пари криптографічних ключів з ідентифікаційними даними, такими як веб-сайти або організації.

Сертифікація X.509 визначає стандартні формати даних і процедури розподілу відкритих ключів за допомогою відповідних сертифікатів з цифровими підписами. Ці сертифікати надаються засвідчують центрами. Крім того, X.509 визначає формат списку анульованих сертифікатів, формат сертифікатів атрибутів і алгоритм перевірки підписи шляхом побудови шляху сертифікації. X.509 припускає наявність ієрархічної системи засвідчуваних центрів для видачі сертифікатів.

Загальні застосування сертифікатів X.509 включають в себе:

- SSL / TLS і HTTPS для аутентифікаційного і зашифрованого перегляду веб-сторінок;
- підписаний і зашифрований адресу електронної пошти через S / MIME протокол;
- підписання коду;
- підписання документів;
- аутентифікація клієнта.

Взагалі по мірі розвитку телекомунікаційних технологій, зокрема, мережі Інтернет, питання забезпечення безпечного доступу користувачів до інформаційних ресурсів стало дуже актуальним, що дозволяє уникнути криптографічної атаки. Одним із способів рішення цього питання, є механізм сертифікації учасників інформаційного обміну. З метою забезпечення сумісності різних апаратів (засобів криптографічного захисту інформації), що використовуються в таких прикладних сервісах (WWW, електронна пошта,

системи аутентифікації користувачів та ін.) був створений стандарт, що описує формат сертифікатів, який був вперше опублікований ІТУ-Т в 1988 році в якості частини рекомендацій X.500 (RFC 5280).

Це був X.509 v1. У 1993 році в опис стандарту сертифікатів додали ще 2 поля, що призвело до появи X.509 v2, а ще трохи пізніше з'явилася третя версія стандарту - X.509 v3. Питаннями реалізації стандарту в мережі Інтернет займається робоча група ІETF, сформована в 1995 році і більш відома, як Public-Key Infrastructure Working Group. Результатами її роботи, зокрема, стали рекомендації RFC 3280 і 5280.

X.509 став основою для побудови ієрархічної системи засвідчуваних центрів, незважаючи на появу в 1991 році технології PGP. Стандарт підтримується в таких протоколах і механізмах як:

- TLS / SSL;
- HTTPS;
- IPsec;
- SSH та інших.

Згідно RFC 1422, стандарт X.509 визначає поняття «сертифікат з відкритим ключем» та інші базові визначення PKIX. При цьому сертифікат з відкритим ключем є певною структурою даних, яка містить:

- ім'я користувача;
- відкриту складову ключа двоключової криптосистеми цього користувача;
- ім'я компанії, яке підтверджує, що відкрита складова прив'язана до імені користувача.

Ці дані через кожен часовий інтервал підписуються організацією. У сертифікатах імена суб'єкта і видавця є помітними іменами, як визначено в системному каталозі X.500.

Після підписання, сертифікати можуть зберігатися на LDAP-серверах. Передаються сертифікати через незахищені обмінні повідомлення або через будь-який інший спосіб, яке робить сертифікати легко доступними при відправці повідомлень користувачам. Сертифікати використовуються в PEM-

- сертифікат;
- алгоритм підпису сертифіката;
- підпис сертифікату.

У 4 розділі RFC 2459 описана структура сертифіката X.509 v3. Оновлена структура сертифіката X.509 v3 описана в документі RFC 5280 в 4 розділі. Вона спеціалізована під інтернет-додатки.

Для опису внутрішньої структури сертифікатів X.509 використовується ASN.1. Зберігаються, як правило, у вигляді DER або PEM-файлів.

Для випуску сертифікатів існує чітко визначена ієрархічна система засвідчуваних центрів, згідно RFC 5280. У цьому його відмінність від моделей, заснованих на принципі мережі довіри, подібної криптографічного технології PGP, де будь-який може випускати, підписувати і перевіряти на відповідність сертифікат. Згідно RFC 4158, X.509 v3 має гнучкість для підтримки таких топологій як мости і сітки, а також може бути використаний в p2p мережах.

Існують обмеження стандарту X.509 при використанні його для шифрування повідомлень електронної пошти та спільної роботи додатків. X.509 пропонує неповну сумісність роботи додатками. А значить, можуть виникнути проблеми при роботі з API. Згідно RFC 4387, суть PKI наступна:

- в браузері зберігаються не сертифікати сайтів, а сертифікати УЦ.

Це означає, що коли ваш браузер отримує сертифікат сайту під час активного з'єднання, він бачить крім адреси сайту ще й «адреса» УЦ, а також цифровий підпис, яку згенерував УЦ з використанням свого секретного ключа. Далі браузер бере з локального сховища ланцюг сертифікатів УЦ, дістає з нього публічний ключ і за допомогою нього перевіряє підпис у сертифікаті сайту. Якщо підпис правильний, з'єднання успішно встановлюється.

Згідно RFC 5280, ланцюг сертифікатів є список сертифікатів (починаючи з особи, посвідченого в якості сервера), що включає один або кілька УЦ (останній підписується самим собою), має такі властивості:

- підпис кожного сертифіката (за винятком останнього) є документом з центром-наступником;
- всі сертифікати, за винятком останнього, підписуються за допомогою закритого ключа наступного УЦ;
- останній сертифікат в ланцюзі є сертифікатом єдиного пункту довіри, що є кореневим УЦ.

Ієрархічні PKI здатні швидко реагувати на компрометацію окремого УЦ усередині інфраструктури. Якщо УЦ скомпрометований, вищестоящий УЦ просто анулює його сертифікат. Як тільки робота УЦ відновлюється, він випускає нові сертифікати для всіх своїх користувачів. Вищий УЦ випускає для скомпрометованого УЦ новий сертифікат з новим відкритим ключем, що дозволяє повернути цей центр назад в ієрархію.

Загальновживані розширення файлів сертифікатів:

- .CER - сертифікат, або набір сертифікатів, закодованих за стандартом CER;
- .DER - сертифікат, закодований за стандартом DER;
- .PEM - DER-сертифікат, закодований в Base64 і поміщений між «-----BEGIN CERTIFICATE -----» і «-----END CERTIFICATE -----»;
- .P7B, .P7C - PKCS#7 містить кілька сертифікатів або CRL.
- .P12 - PKCS#12 містить блок, який зберігає одночасно і закритий ключ, і сертифікат (у зашифрованому вигляді);
- .PFX - PFX, попередник PKCS#12, також містить блок закритого ключа і сертифікат.

2.2 Розробка методики безпечного підключення пристроїв IoT з використанням Служби підготовки пристроїв DPS

Повсюдне впровадження систем Інтернету речей (Internet of Things, IoT) ставлять перед розробниками нові проблеми безпеки.

Безпека IoT - це велика та серйозна тема, однак вона стає більш логічною та зрозумілою, ніж може здатися, у разі виконання конкретних проектів з використанням послуг хмарних середовищ. Хмарні середовища, наприклад такі як Microsoft Azure, надають ряд переваг та можливостей щодо забезпечення безпеки пристроїв, що підключаються. Особливо помітні переваги при підключенні до хмари великої кількості однотипних пристроїв.

Нехай за умовами ведення бізнесу, потрібно підключити досить велику кількість датчиків IoT до хмарного середовища Azure. Скористаємося для цього такими службами, як:

- центр інтернету речей Azure;
- служба підготовки пристроїв Інтернету речей.

Центр Інтернету речей Azure може управляти великими обсягами даних телеметрії, відправлених з багатьох датчиків. Кожен пристрій можна налаштувати окремо, щоб бути впевненим в його справжності. Проте при використанні багатьох пристроїв ця задача буде як мінімум обтяжлива. Процес перевірки автентичності пристрою називається "підготовка". Тому і необхідно застосовувати «Службу підготовки пристроїв Інтернету речей (DPS)», яка забезпечує майже автоматичну підготовку будь-якого числа пристроїв.

Тоді сценарій безпечного підключення датчиків IoT до хмари можна подати у вигляді наступних кроків:

1. Створення ресурсу користувальницького Центру Інтернету речей Azure.
2. Підготовка пристроїв до сертифікації X.509 і реєстрації. В основі сертифікатів X.509 лежить концепція шифрування з відкритим і закритим ключами. При використанні цієї сертифікації зі службою підготовки пристроїв Azure (DPS) інфраструктура відкритих ключів вбудована в службу, що значно спрощує роботу. Служба підготовки пристроїв Azure може бути пов'язана з одним центром Інтернету речей або декількома і може розглядатися як система управління сертифікатами та реєстрацією.
3. Створення Служби підготовки пристроїв, кореневого сертифіката і групової реєстрації. DPS може бути пов'язана з одним центром або декількома. Тобто, це

окремий ресурс, який не залежить від конкретного Центру Інтернету речей. Ресурс DPS створюється так само, як і будь-який інший ресурс Azure. Як політику доступу доцільно обирати iothubowner.

4. Створення кореневого сертифіката X.509 за допомогою Служби підготовки пристроїв. Перш ніж цей сертифікат ЦС можна буде використовувати для перевірки справжності пристроїв в DPS, необхідно провести Підтвердження належності сертифіката.

5. Підтвердження належності сертифіката ЦС надається службі DPS шляхом відправки сертифіката перевірки, створеного на основі кореневого сертифіката. Сертифікат перевірки містить створений код перевірки. Всі ці сертифікати будуть самозавіряємі.

6. Створення групової реєстрації в службі DPS в Azure. На цьому кроці створюємо один кінцевий сертифікат для кожного пристрою, дані телеметрії з якого планується відправляти в центр.

7. Створення коду для пристроїв датчиків. Обиравши інтегроване середовище розробки (IDE) свого додатку та створюємо код, що виконується на кожному пристрої датчика. Одна з цілей полягає в тому, щоб зробити код для кожного пристрою максимально ідентичним і тим самим звести до мінімуму участь людини. Для коду потрібно одне коригування при підготовці для другого пристрою: шлях до кінцевого сертифікату повинен бути унікальним для кожного пристрою. Унікальність цього випадку означає зміну "1" на "2" або "3".

8. Тестування автоматичної підготовки та призначення в Центр для декількох пристроїв. На цьому кроці перевіряється, чи працюють всі датчики належним чином. Спочатку ми перевіряємо, чи отримує центр Інтернету речей автоматично оновлення зі списку підключених пристроїв IoT. Далі ми перевіряємо, чи отримує центр дані телеметрії з пристроїв. Потім, на порталі змінюємо параметр двійника для одного пристрою і переконуємося, що зміни приходять на потрібний пристрій.

2.3. Створення служби підготовки пристроїв в центрі Інтернету речей Azure та кореневого сертифікату

Перший крок - Створення служби підготовки пристроїв в центрі Інтернету речей Azure:

- Главная > Новый > Служба подготовки устройств к добавлению в Центр Интернета вещей

Служба подготовки устройств к добавлению в Центр Интернета вещей

Майкрософт



Служба подготовки устройств к добавл...

Майкрософт

Создать

Сохранить на будущее

Общие сведения

Планы

Простая и автоматизированная регистрация устройств в Центре Интернета вещей и безопасность, которая начинается...

Мы обеспечили подготовку и интеграцию Интернета вещей во всех решениях — от подключенных фабрик до интелл...

Полезные ссылки

 - Документация
 - Дополнительные сведения о службе подготовки устройств к добавлению в Центр Интернета вещей
 - Основные понятия подготовки устройств
 - Сведения о ценах и масштабировании

Рисунок 2.2 - Знімок екрану, який показує вибір служби підготовки пристроїв до додаванню в Центр Інтернету речей

3. Вказуємо ім'я ресурсу, наприклад "CheeseCave-DPS". Використовуємо підписку- "пісочницю" та групу ресурсів. Вибираємо своє місцезнаходження.

Главная > Новый > Служба подготовки устройств... > Служба подготовки устройств к добав...

Служба подготовки устройств к добавлению в Центр Интернета... □ ×

Майкрософт

Имя *
CheeseCave-DPS ✓

Подписка *
Подписка Concierge ▾

Группа ресурсов *
learn-1e8072e2-b583-4715-a0cc-c8667d289a63 ▾
Создать

Расположение * ⓘ
Западная часть США 2 ▾

Создать Параметры автоматизации

Рисунок 2.3 - Знімок екрану, який показує заповнені поля для нової служби підготовки пристроїв

4. Створюємо ресурс та чекаємо його розгортання. Відображення повідомлення про розгортання може зайняти близько двох хвилин.

5. Натискаємо на «Перейти до ресурсу».

Зв'язування ресурсу DPS з Центром Інтернету речей.

1. У ресурсі DPS потрібно знайти та вибрати «Пов'язані центри Інтернету речей» в розділі «Параметри» в меню зліва.

2. Вибираємо «Додати».

3. В області «Додати посилання на центр Інтернету речей» виберіть підписку- «пісочницю».

4. Знаходимо «ім'я свого центру Інтернету речей» та вибираємо «iothubowner», як «Політики доступу». Натискаємо «Зберегти».

Добавить ссылку на центр Интернета вещей

[Дополнительные сведения о связывании центров Интернета вещей.](#)

Подписка * ⓘ
Подписка Concierge

Центр Интернета вещей *
CheeseCaveHub

Политика доступа * ⓘ
iothubowner

Имя узла: CheeseCaveHub.azure-devices.net

Состояние: Активный

Ценовая категория: S1

Расположение: Западная часть США 2

Сохранить

Рисунок 2.4 - Знімок екрану, який показує, як зв'язати центр Інтернету речей з DPS

5. Натискаємо «Управління політикою розподілу» та переконуємося, що для політики задано «Рівномірно виважений розподіл».

CheeseCave-DPS — Управление политикой выделения
Служба подготовки устройств

Поиск (CTRL+J) Сохранить

Обзор
Журнал действий
Управление доступом (IAM)
Теги
Диагностика и решение проблем

Параметры
Быстрое начало
Политики общего доступа
Связанные центры Интернета...
Certificates
Управление регистрациями
Управление политикой выдел...
Properties
Блокировки
Экспорт шаблона

Наименьшая задержка: эта политика назначает устройства в связанные Центры Интернета вещей так, чтобы задержка при обмене данными между устройствами и Центрами Интернета вещей была н. Равномерно взвешенное распределение: эта политика равномерно распределяет устройства между Центрами Интернета вещей. Мы рекомендуем использовать эту политику, если вы развертываете. Статическая конфигурация: эта политика требует, чтобы необходимый Центр Интернета вещей указывался в записи регистрации дляготавливаемого устройства.

Выберите способ назначения устройств в центрах

- ☐ Наименьшая задержка
- ☒ Равномерное взвешенное распределение
- ☐ Статическая конфигурация (с помощью только списка регистраций)

Рисунок 2.5 - Знімок екрану, який показує політики розподілу DPS

6. На сторінці «Огляд» копіюємо рядок «Область ідентифікатора» в текстовий файл. Рядок відображається в правому верхньому кутку сторінки.

7. Також можна додати закладку на сторінку «Огляд цього ресурсу».

Перш ніж перейти далі в роботі з ресурсом DPS та почати додавати реєстрації, необхідно спочатку створити кореневий сертифікат.

Створення кореневого сертифікату X.509. При першому створенні всіх сертифікатів X.509 необхідно завантажити певні інструменти.

1. Натискаємо правою кнопкою миші «Microsoft Azure Cloud Shell» та вибираємо «Відкрити в новому вікні».

2. Переконаємось, що ми обрали оболонки «Bash». Цей параметр необхідний, так як допоміжні засоби, які будуть завантажені далі, написані для оболонки «Bash».

3. Запускаємо зазначений скрипт. Він створює каталог сертифікатів в сховище оболонки та завантажує в нього допоміжні скрипти.

Скрипт:

```
# create certificates directory
mkdir certificates

# navigate to certificates directory
cd certificates

# download helper script files
curl https://raw.githubusercontent.com/Azure/azure-iot-sdk-c/master/tools/CACertificates/certGen.sh --output certGen.sh
curl https://raw.githubusercontent.com/Azure/azure-iot-sdk-c/master/tools/CACertificates/openssl_device_intermediate_ca.cnf --output openssl_device_intermediate_ca.cnf
curl https://raw.githubusercontent.com/Azure/azure-iot-sdk-c/master/tools/CACertificates/openssl_root_ca.cnf --output openssl_root_ca.cnf
# update script permissions so user can read, write, and execute it
chmod 700 certGen.sh
```

Ці допоміжні скрипти завантажуються з проекту «Azure / azure-iot-sdk-c» з відкритим вихідним кодом, розміщеного на сайті «GitHub». Він є частиною пакета «SDK» для «Azure IoT». Допоміжний скрипт «certGen.sh» допоможе продемонструвати призначення сертифікатів ЦС, не заглиблюючись в особливості конфігурації «OpenSSL».

4. Виконуємо наступну команду, щоб створити кореневий та проміжний сертифікати. Нам потрібен лише кореневий сертифікат, який буде називатися «azure-iot-test-only.root.ca.cert.pem».

Лістинг коду:

```
./certGen.sh create_root_and_intermediate
```

5. Необхідно завантажити кореневий сертифікат на локальний комп'ютер, а потім відправити його в службу «DPS» в «Azure». Вводимо:

Лістинг коду:

```
download ~/certificates/certs/azure-iot-test-only.root.ca.cert.pem
```

6. Створюємо в папці «Документи» нову папку з ім'ям «cheese seva certs» або схожим ім'ям.

7. Копіюємо завантажений файл сертифікату в папку «cheese seva certs».

Налаштування відносини довіри з кореневим сертифікатом для служби DPS в Azure:

1. На порталі Azure переходимо до ресурсу DPS cheese-cave, який був створений раніше.

2. В розділі «Параметри» натискаємо «Сертифікати».

3. Натискаємо «Додати». В полі «Ім'я сертифікату» вводимо зрозуміле ім'я, наприклад «cheesecave-dps-root». Ім'я не повинно збігатись з ім'ям файлу сертифікату.

4. В полі «PEM» або «CER-файл сертифікату» переходимо в папку «cheese seva certs» та вибираємо завантажений файл «azure-iot-test-only.root.ca.cert.pem».

5. Натискаємо «Зберегти».

Після завантаження кореневого сертифікату він відобразиться в області «Сертифікати» зі станом «Неперевірений». Перш ніж цей сертифікат ЦС можна буде використовувати для перевірки справжності пристроїв в «DPS», необхідно провести «Підтвердження належності сертифіката».

Підтвердження належності:

1. В області «Сертифікати» натискаємо «сертифікат», щоб відкрити «Відомості про сертифікат».
2. Натискаємо «Створити код перевірки».
3. Копіюємо «код перевірки», який відображається над кнопкою «Створити код перевірки». Кнопка справа від текстового поля слугує для копіювання коду. Відкриваємо текстовий редактор, наприклад «Блокнот» та вставляємо «код перевірки».

Підтвердження належності сертифікату ЦС надається службі «DPS» шляхом відправки сертифікату перевірки, створеного на основі кореневого сертифікату. Сертифікат перевірки містить щойно створений «код перевірки».

4. При створенні сертифіката перевірки не закриваємо панель «Відомості про сертифікат». Якщо закрити панель, код перевірки стане недійсним і буде потрібно створити новий.
5. Копіюємо наступну команду в текстовий файл, що містить код перевірки, і змініть «<verification-code>» на фактичний код.

Команда:

```
./certGen.sh create_verification_certificate <verification-code>
```

6. Повертаємось в «Cloud Shell» і в каталозі «certificates» виконуємо готову команду. Ця команда створює сертифікат перевірки, пов'язаний з кореневим сертифікатом. Створений сертифікат називається «verification-code.cert.pem» і знаходиться в каталозі «./certs» в «Cloud Shell».
7. Виконуємо наступну команду, щоб завантажити сертифікат перевірки на локальний комп'ютер.

Команда:

```
download ~/certificates/certs/verification-code.cert.pem
```

8. Копіюємо завантажений файл в папку «cheese cave certs».

9. Повертаємось до області «Відомості про сертифікат порталу Azure». В полі «PEM» або «CER-файл сертифіката перевірки» вибираємо завантажений «verification-code.cert.pem».

10. Натискаємо «Перевірити» і в області «Сертифікати» перевіряємо стан сертифіката, яке тепер відображається, як «Перевірено». Щоб побачити цю зміну, може знадобитися кнопка "Оновити" у верхній частині панелі.

Тепер ми створили кореневий сертифікат та перевірили його в службі «DPS» в Azure. Наступний наш шаг – створити «групову реєстрацію» та зв'язати її з кореневим сертифікатом. Створюємо групову реєстрацію в службі «DPS» в Azure:

1. На порталі Azure з обраним ресурсом «DPS» відкриваємо розділ «Параметри».

2. Натискаємо «Управління реєстраціями», а потім «Додати групу реєстрації». А також потрібно переконатись, що ми додаємо групу, а не окрему реєстрацію.

3. В поле «Ім'я групи» вводимо "cheesecave-devices".

4. Переконуємось, що в полі «Тип атестації» задано значення «Сертифікат», а в полі «Тип сертифіката» - «сертифікат ЦС». Це параметри за замовчуванням.

5. В списку «Основний сертифікат» вибираємо «сертифікат ЦС». Це буде «cheesecave-dps-root», якщо ми використовували запропоноване ім'я.

6. Залишаємо в списку «Вторинний сертифікат» значення «Сертифікат не вибрано». Нам не обов'язково ставити вторинний сертифікат.

7. Залишаємо в полі «Виберіть спосіб призначення пристроїв в центрах» значення «Рівномірно виважений розподіл». Так як з реєстрацією пов'язаний тільки один центр «Інтернету речей», цей параметр не важливий.

8. Переконуємось, що наш центр «Інтернету речей» обраний в полі «Виберіть центри Інтернету речей», яким може бути призначено пристрій. Для параметрів повторної підготовки залишаємо значення за замовчуванням.

9. Натискаємо кнопку «Зберегти» та переконуємось, що нова група реєстрації з'явилася в області «Групи реєстрації».

2.4. Створення декількох кінцевих сертифікатів

На цьому етапі нам потрібно створити один кінцевий сертифікат для кожного пристрою, дані телеметрії з якого повинні відправлятися в центр. Створюємо код для великої кількості пристроїв. Згадаємо, що в описі сценарію ми вказали, що у нас 30 сироварень, для яких потрібно 30 пристроїв датчиків. Знизимо це число до трьох для простоти пояснення та тестування:

1. У Microsoft Azure Cloud Shell переконуємось, що ми перебуваємо в папці «certificates».

2. Виконуємо наступну команду:

```
./certGen.sh create_device_certificate cheesecake-device1
```

3. Переконуємось, що кінцевий сертифікат створений правильно. Ми повинні побачити в консолі безліч повідомлень, що закінчуються приблизно наступним чином:

```
Signature Algorithm: sha256WithRSAEncryption
cb:bb:4b:34:8e:76:97:bc:36:ee:74:c4:5c:94:7c:91:2d:aa:
63:22:6a:69:05:cc:3b:86:c5:a4:18:92:80:73:21:dd:be:16:
50:42:b7:4d:4c:5b:f9:34:61:28:33:f4:fc:8f:57:27:45:3f:
d8:c2:7b:61:d1:a6:8a:66:4b:c5:f0:af:11:79:a7:aa:4b:2f:
1c:59:73:68:67:fb:7e:bd:21:06:1d:4f:20:63:f2:62:3e:62:
b1:ba:90:f9:1c:52:23:07:4c:f3:9f:fb:3b:7b:ef:b0:69:cc:
7b:a3:17:c4:40:4c:93:59:e9:a2:2c:aa:b3:58:13:df:52:b8:
7d:e5:e4:39:7e:d6:3d:02:ea:87:f8:3d:21:e1:e3:7e:72:1c:
f1:bd:f0:fe:19:17:fe:00:3f:8c:b2:67:a7:09:97:2a:dd:70:
ae:b0:1f:1d:ae:92:19:52:fa:10:cb:23:e8:1b:70:48:0c:8b:
cb:03:f9:55:cd:4e:f5:c1:7c:b8:76:41:8e:b4:bd:5a:bc:32:
b4:8c:c4:26:36:4f:45:92:fa:44:21:a6:57:ad:39:d5:c1:d4:
ce:ee:03:c9:ed:81:36:0d:a4:ca:17:bd:3d:34:5c:b2:18:0a:
69:19:5f:45:31:55:39:c5:dc:33:6a:f5:4d:3e:ee:0d:14:40:
3b:49:ba:42:21:7b:e2:28:50:2e:34:3c:fa:97:b4:5f:95:4b:
c1:0d:0f:f5:cb:0d:56:f0:65:90:c7:69:8c:e6:b9:cd:74:25:
07:6c:02:8a:b1:46:11:8e:1e:7e:df:fc:cd:cb:49:11:ed:8d:
65:1f:25:35:c4:16:df:06:5f:9e:0a:95:da:76:f6:04:5b:89:
f9:09:54:bc:3f:54:7d:0d:08:85:64:d1:3b:57:cc:72:3e:ab:
32:da:8f:f0:6a:b5:d5:3c:21:52:85:6b:aa:71:66:33:9f:a5:
04:d3:c2:92:ce:48:d4:95:b6:c0:e3:5d:5b:b8:65:f6:ab:20:
b9:e5:10:5b:54:73:45:b0:ca:fc:23:98:ae:5c:a1:b9:3b:64:
c8:27:73:b2:0f:38:57:5c:45:34:e2:b7:3f:d1:29:84:94:b4:
cd:31:da:01:eb:fb:61:2e:fe:2d:60:09:45:ff:11:55:aa:9e:
09:68:ae:f8:3e:11:3f:d9:20:75:1d:12:b9:38:34:33:7b:df:
a6:d9:62:47:25:2b:98:c4:35:8e:09:a8:14:c0:9d:31:a1:9e:
c4:00:6d:4d:55:e9:8c:d0:7b:ef:5e:57:b9:0d:8c:c7:04:40:
fb:aa:ed:b1:aa:1c:33:5d:51:df:98:a3:a2:17:58:25:c3:f8:
a9:be:24:39:2f:60:1d:12
Create the Leaf Device PFX Certificate
-----
Leaf Device PFX Certificate Generated At:
-----
./certs/new-device.cert.pfx
Certs generated by this script are not for production (e.g. they have hard-coded passwords of '1234').
This script is only to help you understand Azure IoT Hub CA Certificates.
Use your official, secure mechanisms for this cert generation.
Also note that these certs will expire in 30 days.
```

Рисунок 2.6 - Вихідні дані консолі, що описують правильне створення кінцевого сертифіката

4. Щоб завантажити створений сертифікат на локальний комп'ютер, вводимо:
download ~/certificates/certs/new-device.cert.pfx

5. Копіюємо файл «new-device.cert.pfx» з розташування скачування в папку «Downloads», в папку «cheese cave certs» та перейменуємо його в «new-device.cert1.pfx».

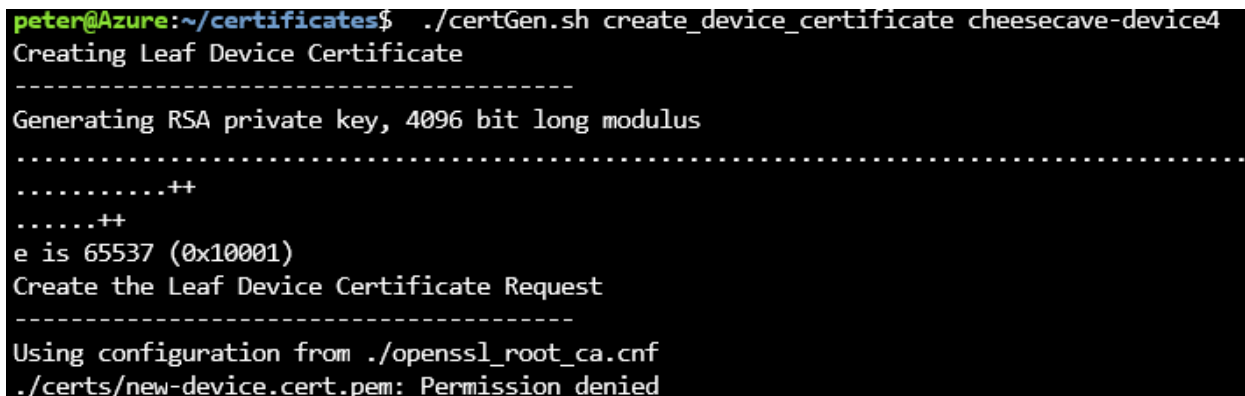
6. У «Cloud Shell» видаляємо файли «new-device.cert». Цей крок потрібен, щоб ті ж кошти можна було використати для створення подальших сертифікатів. Ми можемо видалити цей файл в папці «certificates / certs», виконавши наступні кроки:

- ввести «cd certs»;
- ввести «rm new *»;
- відповісти «у» на питання щодо захищеного файлу;
- ввести «cd».

7. У «Cloud Shell» створюємо другий кінцевий сертифікат:

```
./certGen.sh create_device_certificate cheesecave-device2
```

8. Якщо ми отримаємо відповідь «Permission denied», то потрібно переконатись, що попередні сертифікати були вилучені належним чином.



```
peter@Azure:~/certificates$ ./certGen.sh create_device_certificate cheesecave-device4
Creating Leaf Device Certificate
-----
Generating RSA private key, 4096 bit long modulus
.....++
.....++
e is 65537 (0x10001)
Create the Leaf Device Certificate Request
-----
Using configuration from ./openssl_root_ca.cnf
./certs/new-device.cert.pem: Permission denied
```

Рисунок 2.7 - Вихідні дані консолі, що вказують, що кінцевий сертифікат не створено

9. Щоб завантажити другий сертифікат на локальний комп'ютер, вводимо:

```
download ~/certificates/certs/new-device.cert.pfx
```

10. Копіюємо файл «new-device.cert.pfx» з розташування скачування в папку «cheese cave certs» та перейменуємо його в «new-device.cert2.pfx».

11. У Cloud Shell видаляємо файли «new-device.cert» та виконуємо наступні кроки:

- вводимо «cd certs»;
- вводимо «rm new*»;
- відповідаємо «у» на питання, щодо захищеного файлу;
- вводимо «cd».

12. У «Cloud Shell» створюємо третій кінцевий сертифікат:

```
./certGen.sh create_device_certificate cheesecave-device3
```

13. Щоб завантажити третій сертифікат на локальний комп'ютер, вводимо:

```
download ~/certificates/certs/new-device.cert.pfx
```

14. Копіюємо файл «new-device.cert.pfx» з розташування скачування в папку «cheese cave certs» та перейменуємо його в «new-device.cert3.pfx».

Висновок до 2 розділу:

Другий розділ був посвячений розробці та створенню методики безпечного підключення пристроїв. Перш ніж почавши розробку підключення, ми провели детальний аналіз та дали розгорнутий опис, щодо особливостей сертифікації X.509. Визначили, що «Сертифікат X.509» надійно пов'язує пари криптографічних ключів з ідентифікаційними даними. Визначали що саме він надає користувачам, які є різновиди цього сертифікату, в яких протоколах та механізмах він підтримується, навели приклад загального стандарту для інтернету, який використовує формат X.509, у вигляді картинки та трішки торкнулися історії виникнення цього сертифікату.

Затім ми розробили методику безпечного підключення пристроїв з використанням служби підготовки пристроїв DPS. Визначили основні поняття, щодо служби підготовки пристроїв. Для чого вона потрібна, які переваги надає

своїм користувачам та пристроям, які підключають. Як служба DPS пов'язана с сертифікатом X.509.

Та виділили сценарій з переліком певних кроків, яким потрібно слідувати, щоб правильно створити безпечне підключення пристроїв.

Завершуючи цей розділ, ми перейшли від теорії до практики та створили службу підготовки пристроїв в центрі Інтернету речей Azure та кореневого сертифікату. Результати нашої роботи показані у вигляді скрінів та рисунків.

РОЗДІЛ 3. ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ ТА ЕФЕКТИВНОСТІ ПРИЙНЯТИХ РІШЕНЬ

3.1 Розробка коду для пристроїв датчиків

У цьому пункті 3 розділу ми створюємо код, що виконується на кожному пристрою датчика. Одна із цілей полягає в тому, щоб створити код для кожного пристрою максимально ідентичним та тим самим звести до мінімуму участь людини. Для цього коду потрібно одне коригування при підготовці для другого пристрою - шлях до кінцевого сертифікату повинен бути унікальним для кожного пристрою.

Унікальність цього випадку означає зміну "1" на "2" або "3". Ця зміна настільки незначна, що його можна легко автоматизувати в робочому середовищі.

Код написаний на мові «C#». В якості середовища розробки можна вибрати «Visual Studio» або «Visual Studio Code».

Для створення проекту для імітації одного пристрою, потрібно виконати наступні кроки:

1. Для використання «C#» в «Visual Studio Code» повинні бути встановлені «.NET Core» та розширення «C#».
2. Потрібно відкрити термінал в «Visual Studio Code» та створити папку з ім'ям "CaveDevice1". Далі потрібно перейти в папку «CaveDevice1».
3. Вводимо у вікні терміналу наступну команду:
- dotnet new console.
Ця команда створює в папці файл «Program.cs», а також файл проекту.
4. Вводимо в терміналі «dotnet restore». Ця команда надає додаткам доступ до необхідних пакетів «.NET».
5. У терміналі нам потрібно встановити певні необхідні бібліотеки. Щоб це зробити нам потрібно ввести наступні команди:
- dotnet add package Microsoft.Azure.Devices.Client;

- dotnet add package Microsoft.Azure.Devices.Provisioning.Client;
- dotnet add package Microsoft.Azure.Devices.Provisioning.Transport.Amqp.

6. З меню «Файл» відкриваємо файл «Program.cs» та видаляємо вміст за замовчуванням. Переходимо для цього в папку «CaveDevice1».

7. Після введення наведеного нижче коду в файл «Program.cs» можна запустити програму за допомогою команди «dotnet run». Ця команда запустить файл «Program.cs» в цій папці. Тому переконуємось в тому, що ми перебуваємо в папці «CaveDevice1».

Для додавання коду для відправки телеметрії виконуємо наступні кроки:

1. Відкриваємо файл «Program.cs» для додатка пристрої.
2. Вставляємо наступний код.

Лістинг коду:

```
// Copyright (c) Microsoft. All rights reserved.
```

```
// Licensed under the MIT license. See LICENSE file in the project root for full license information.
```

```
using Microsoft.Azure.Devices.Client;
using Microsoft.Azure.Devices.Provisioning.Client;
using Microsoft.Azure.Devices.Provisioning.Client.Transport;
using Microsoft.Azure.Devices.Shared;
using System;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using Newtonsoft.Json;
using System.Security.Cryptography.X509Certificates;
```

```
namespace X509CertificateSimulatedDevice
{
    class Program
```

```

{
    // Azure Device Provisioning Service (DPS) Global Device Endpoint.
    private const string GlobalDeviceEndpoint = "global.azure-devices-
provisioning.net";

    // Azure Device Provisioning Service (DPS) ID Scope.
    private static string dpsIdScope = "<your DPS ID Scope>";

    // Certificate (PFX) File Name.
    private static string s_certificateFileName = "<your leaf path>";

    // Certificate (PFX) Password. Better to use a Hardware Security Module for
production devices.
    private static string s_certificatePassword = "1234";
    public static int Main(string[] args)
    {
        X509Certificate2 certificate = LoadProvisioningCertificate();
        using (var security = new SecurityProviderX509Certificate(certificate))
        {
            using (var transport = new
ProvisioningTransportHandlerAmqp(TransportFallbackType.TcpOnly))
            {
                ProvisioningDeviceClient provClient =
                    ProvisioningDeviceClient.Create(GlobalDeviceEndpoint, dpsIdScope,
security, transport);
                var provisioningDeviceLogic = new
ProvisioningDeviceLogic(provClient, security);
                provisioningDeviceLogic.RunAsync().GetAwaiter().GetResult();
            }
        }
    }
}

```

```

        return 0;
    }
    private static X509Certificate2 LoadProvisioningCertificate()
    {
        var certificateCollection = new X509Certificate2Collection();
        certificateCollection.Import(s_certificateFileName, s_certificatePassword,
X509KeyStorageFlags.UserKeySet);
        X509Certificate2 certificate = null;
        foreach (X509Certificate2 element in certificateCollection)
        {
            Console.WriteLine($"Found certificate: {element?.Thumbprint}
{element?.Subject}; PrivateKey: {element?.HasPrivateKey}");
            if (certificate == null && element.HasPrivateKey)
            {
                certificate = element;
            }
            else
            {
                element.Dispose();
            }
        }
        if (certificate == null)
        {
            throw new FileNotFoundException($"{{s_certificateFileName}} did not
contain any certificate with a private key.");
        }
        Console.WriteLine($"Using certificate {certificate.Thumbprint}
{certificate.Subject}");
        return certificate;
    }

```

```
}
```

```
// The ProvisioningDeviceLogic class contains the device logic to read from the  
simulated Device Sensors, and send Device-to-Cloud
```

```
// messages to the Azure IoT Hub. It also contains the code that updates the device  
with changes to the device twin properties.
```

```
public class ProvisioningDeviceLogic
```

```
{
```

```
    readonly ProvisioningDeviceClient _provClient;
```

```
    readonly SecurityProvider _security;
```

```
    DeviceClient s_deviceClient;
```

```
    // Global constants.
```

```
    const float ambientTemperature = 70;                // Ambient temperature of a  
southern cave, in degrees F.
```

```
    const double ambientHumidity = 99;                // Ambient humidity in  
relative percentage of air saturation.
```

```
    const double desiredTempLimit = 5;                // Acceptable range above or  
below the desired temp, in degrees F.
```

```
    const double desiredHumidityLimit = 10;           // Acceptable range above or  
below the desired humidity, in percentages.
```

```
    const int intervalInMilliseconds = 5000;         // Interval at which telemetry  
is sent to the cloud.
```

```
    enum stateEnum
```

```
    {
```

```
        off,
```

```
        on,
```

```
        failed
```

```
    }
```

```
    // Global variables.
```

```

        private static stateEnum fanState = stateEnum.off;           // Initial setting
of the fan.

        private static double desiredTemperature = ambientTemperature - 10;    // Initial
desired temperature, in degrees F.

        private static double desiredHumidity = ambientHumidity - 20;        // Initial
desired humidity in relative percentage of air saturation.

        public ProvisioningDeviceLogic(ProvisioningDeviceClient
provisioningDeviceClient, SecurityProvider security)
        {
            _provClient = provisioningDeviceClient;
            _security = security;
        }

        private static void colorMessage(string text, ConsoleColor clr)
        {
            Console.ForegroundColor = clr;
            Console.WriteLine(text);
            Console.ResetColor();
        }

        private static void greenMessage(string text)
        {
            colorMessage(text, ConsoleColor.Green);
        }

        private static void redMessage(string text)
        {
            colorMessage(text, ConsoleColor.Red);
        }

        private static void whiteMessage(string text)
        {
            colorMessage(text, ConsoleColor.White);
        }

```

```

public async Task RunAsync()
{
    colorMessage($"\\nRegistrationID = {_security.GetRegistrationID()}",
ConsoleColor.Yellow);
    // Register the Device with DPS.
    whiteMessage("ProvisioningClient RegisterAsync . . . ");
    DeviceRegistrationResult result = await
_provClient.RegisterAsync().ConfigureAwait(false);
    if (result.Status == ProvisioningRegistrationStatusType.Assigned)
    {
        greenMessage($"Device Registration Status: {result.Status}");
        greenMessage($"ProvisioningClient AssignedHub: {result.AssignedHub};
DeviceID: {result.DeviceId}");
    }
    else
    {
        redMessage($"Device Registration Status: {result.Status}");
        throw new Exception($"DeviceRegistrationResult.Status is NOT
'Assigned'");
    }
    // Create x509 DeviceClient Authentication.
    whiteMessage("Creating X509 DeviceClient authentication.");
    var auth = new DeviceAuthenticationWithX509Certificate(result.DeviceId,
(_security as SecurityProviderX509).GetAuthenticationCertificate());
    whiteMessage("Simulated Device. Ctrl-C to exit.");
    using (s_deviceClient = DeviceClient.Create(result.AssignedHub, auth,
TransportType.Amqp))
    {
        // Explicitly open DeviceClient to communicate with Azure IoT Hub.
        whiteMessage("DeviceClient OpenAsync.");
    }
}

```

```

        await s_deviceClient.OpenAsync().ConfigureAwait(false);
        // Setup OnDesiredPropertyChanged Event Handling to receive Desired
Properties changes.
        whiteMessage("Connecting SetDesiredPropertyUpdateCallbackAsync
event handler...");
        await
s_deviceClient.SetDesiredPropertyUpdateCallbackAsync(OnDesiredPropertyChange
d, null).ConfigureAwait(false);
        // Load Device Twin Properties since device is just starting up.
        whiteMessage("Loading Device Twin Properties...");
        var twin = await s_deviceClient.GetTwinAsync().ConfigureAwait(false);
        // Use OnDesiredPropertyChanged event handler to set the loaded Device
Twin properties.
        await OnDesiredPropertyChanged(twin.Properties.Desired, null);
        // Start reading and sending device telemetry.
        colorMessage("\nStart reading and sending device telemetry...\n",
ConsoleColor.Yellow);
        await SendDeviceToCloudMessagesAsync2(s_deviceClient);
        // Explicitly close DeviceClient.
        whiteMessage("DeviceClient CloseAsync.");
        await s_deviceClient.CloseAsync().ConfigureAwait(false);
    }
}
private async Task OnDesiredPropertyChanged(TwinCollection
desiredProperties, object userContext)
{
    whiteMessage("Desired Twin Property Changed:");
    whiteMessage($"{desiredProperties.ToJson()}");
    // Read the desired Twin Properties.
    if (desiredProperties.Contains("fanstate") & fanState != stateEnum.failed)

```

```

{
    string desiredFanState = desiredProperties["fanstate"];
    desiredFanState = desiredFanState.ToLower();
    if (desiredFanState == "on" || desiredFanState == "off")
    {
        fanState = (stateEnum)Enum.Parse(typeof(stateEnum), desiredFanState);
        greenMessage($"Set the fan to: {desiredFanState}");
    }
    else
    {
        redMessage($"Illegal fan state received: {desiredFanState}");
    }
}
if (desiredProperties.Contains("temperature"))
{
    string desiredTemperatureString = desiredProperties["temperature"];
    try
    {
        desiredTemperature = double.Parse(desiredTemperatureString);
        greenMessage($"Setting the desired temperature to:
{desiredTemperatureString}");
    }
    catch
    {
        redMessage($"Illegal temperature received:
{desiredTemperatureString}");
    }
}
if (desiredProperties.Contains("humidity"))
{

```

```

        string desiredHumidityString = desiredProperties["humidity"];
    try
    {
        desiredHumidity = double.Parse(desiredHumidityString);
        greenMessage($"Setting the desired humidity to:
{desiredHumidityString}");
    }
    catch
    {
        redMessage($"Illegal humidity received: {desiredHumidityString}");
    }
}
// Report Twin properties.
var reportedProperties = new TwinCollection();
reportedProperties["fanstate"] = fanState.ToString();
reportedProperties["humidity"] = desiredHumidity;
reportedProperties["temperature"] = desiredTemperature;
await
s_deviceClient.UpdateReportedPropertiesAsync(reportedProperties).ConfigureAwait
(false);
    greenMessage($"Reported Twin Properties: {reportedProperties.ToJson()}");
}
// Async method to send simulated telemetry.
private async Task SendDeviceToCloudMessagesAsync2(DeviceClient
deviceClient)
{
    double currentTemperature = ambientTemperature;    // Initial setting of
temperature.
    double currentHumidity = ambientHumidity;          // Initial setting of
humidity.

```

```

    Random rand = new Random();
while (true)
{
    // Simulate telemetry.
    double deltaTemperature = Math.Sign(desiredTemperature -
currentTemperature);
    double deltaHumidity = Math.Sign(desiredHumidity - currentHumidity);
    if (fanState == stateEnum.on)
    {

        // If the fan is on the temperature and humidity will be nudged towards
the desired values most of the time.
        currentTemperature += (deltaTemperature * rand.NextDouble()) +
rand.NextDouble() - 0.5;
        currentHumidity += (deltaHumidity * rand.NextDouble()) +
rand.NextDouble() - 0.5;
        // Randomly fail the fan.
        if (rand.NextDouble() < 0.01)
        {
            fanState = stateEnum.failed;
            redMessage("Fan has failed");
        }
    }
    else
    {
        // If the fan is off, or has failed, the temperature and humidity will creep
up until they reaches ambient values,
        // thereafter fluctuate randomly.
        if (currentTemperature < ambientTemperature - 1)
        {

```

```

        currentTemperature += rand.NextDouble() / 10;
    }
    else
    {
        currentTemperature += rand.NextDouble() - 0.5;
    }
    if (currentHumidity < ambientHumidity - 1)
    {
        currentHumidity += rand.NextDouble() / 10;
    }
    else
    {
        currentHumidity += rand.NextDouble() - 0.5;
    }
}
// Check: humidity can never exceed 100%.
currentHumidity = Math.Min(100, currentHumidity);
// Create JSON message.
var telemetryDataPoint = new
{
    temperature = Math.Round(currentTemperature, 2),
    humidity = Math.Round(currentHumidity, 2)
};
var messageString = JsonConvert.SerializeObject(telemetryDataPoint);
var message = new Message(Encoding.ASCII.GetBytes(messageString));
// Add custom application properties to the message.
message.Properties.Add("sensorID", "S1");
message.Properties.Add("fanAlert", (fanState == stateEnum.failed) ? "true"
: "false");
// Send temperature or humidity alerts, only if they occur.

```

```

        if ((currentTemperature > desiredTemperature + desiredTempLimit) ||
(currentTemperature < desiredTemperature - desiredTempLimit))
        {
            message.Properties.Add("temperatureAlert", "true");
        }
        if ((currentHumidity > desiredHumidity + desiredHumidityLimit) ||
(currentHumidity < desiredHumidity - desiredHumidityLimit))
        {
            message.Properties.Add("humidityAlert", "true");
        }
        colorMessage($"Message data: {messageString}", ConsoleColor.White);
        // Send the telemetry message
        await deviceClient.SendEventAsync(message);
        greenMessage("Message sent\n");
        await Task.Delay(intervalInMilliseconds);
    }
}
}
}

```

3. Змінюємо рядок «your DPS ID Score» на фактичну область ідентифікатора DPS, яка була збережена в попередньому розділі.

4. Змінюємо «your leaf path» на фактичний шлях до кінцевого сертифікату. Це можна зробити, замінивши рядок коду, що містить цей рядок, на наступну інструкцію, яка показана нижче. Змініть «your user name» вашим ім'ям користувача:

```

- private static string s_certificateFileName = "C:\\Users\\<your user
name>\\Documents\\cheese cave certs\\new-device.cert1.pfx";

```

5. Відкриваємо програму. У «Visual Studio Code» вводимо «dotnet run» у вікні терміналу. Потім у «Visual Studio» вибираємо «Налагодження / Запуск без налагодження».

```

C:\WINDOWS\system32\cmd.exe
Found certificate: E6BC58C2D3D8C03CEC2E183117E5F81512329FC1 CN=cheesecave-device1; PrivateKey: True
Using certificate E6BC58C2D3D8C03CEC2E183117E5F81512329FC1 CN=cheesecave-device1

RegistrationID = cheesecave-device1
ProvisioningClient RegisterAsync . . .
Device Registration Status: Assigned
ProvisioningClient AssignedHub: VibrationSensorHub.azure-devices.net; DeviceID: cheesecave-device1
Creating X509 DeviceClient authentication.
Simulated Device. Ctrl-C to exit.
DeviceClient OpenAsync.
Connecting SetDesiredPropertyUpdateCallbackAsync event handler...
Loading Device Twin Properties...
Desired Twin Property Changed:
{"temperature": "55", "version": 1}
Setting the desired temperature to: 55
Reported Twin Properties: {"fanstate": "off", "humidity": 79.0, "temperature": 55.0}

Start reading and sending device telemetry...

Message data: {"temperature": 69.72, "humidity": 99.24}
Message sent
Message data: {"temperature": 70.07, "humidity": 98.77}
Message sent
Message data: {"temperature": 70.54, "humidity": 98.86}
Message sent
  
```

Рисунок 3.1 - Консоль, яка показує, що перший пристрій працює правильно. Для того, щоб створити програму для другого пристрою, виконуємо наступні кроки:

1. Повторюємо всі дії, описані в розділі «Створення проекту для імітації одного пристрою», але нову програму створюємо з новим ім'ям проекту:
- CaveDevice2.
2. Змінюємо ім'я кінцевого сертифіката з «new-device.cert1.pfx» на «new-device.cert2.pfx».
3. Запускаємо програму:

```

C:\WINDOWS\system32\cmd.exe
Found certificate: 830E5BD195F39E4ACDC33C0003A2E8E2F850110A CN=cheesecave-device2; PrivateKey: True
Using certificate 830E5BD195F39E4ACDC33C0003A2E8E2F850110A CN=cheesecave-device2

RegistrationID = cheesecave-device2
ProvisioningClient RegisterAsync . . .
Device Registration Status: Assigned
ProvisioningClient AssignedHub: VibrationSensorHub.azure-devices.net; DeviceID: cheesecave-device2
Creating X509 DeviceClient authentication.
Simulated Device. Ctrl-C to exit.
DeviceClient OpenAsync.
Connecting SetDesiredPropertyUpdateCallbackAsync event handler...
Loading Device Twin Properties...
Desired Twin Property Changed:
{"temperature": "55", "version": 1}
Setting the desired temperature to: 55
Reported Twin Properties: {"fanstate": "off", "humidity": 79.0, "temperature": 55.0}

Start reading and sending device telemetry...

Message data: {"temperature": 69.94, "humidity": 98.73}
Message sent
Message data: {"temperature": 70.41, "humidity": 98.52}
Message sent
Message data: {"temperature": 70.52, "humidity": 98.87}
Message sent
  
```

Рисунок 3.2 - Консоль, яка показує, що другий пристрій працює правильно.

Для створення програми для третього пристрою, виконуємо наступні кроки:

1. Повторюємо всі дії, описані в розділі «Створення проекту для імітації одного пристрою», але нову програму створюємо з новим ім'ям проекту:

- CaveDevice3.

2. Змінюємо ім'я кінцевого сертифіката з «new-device.cert1.pfx» на «new-device.cert3.pfx».

3. Запускаємо програму:

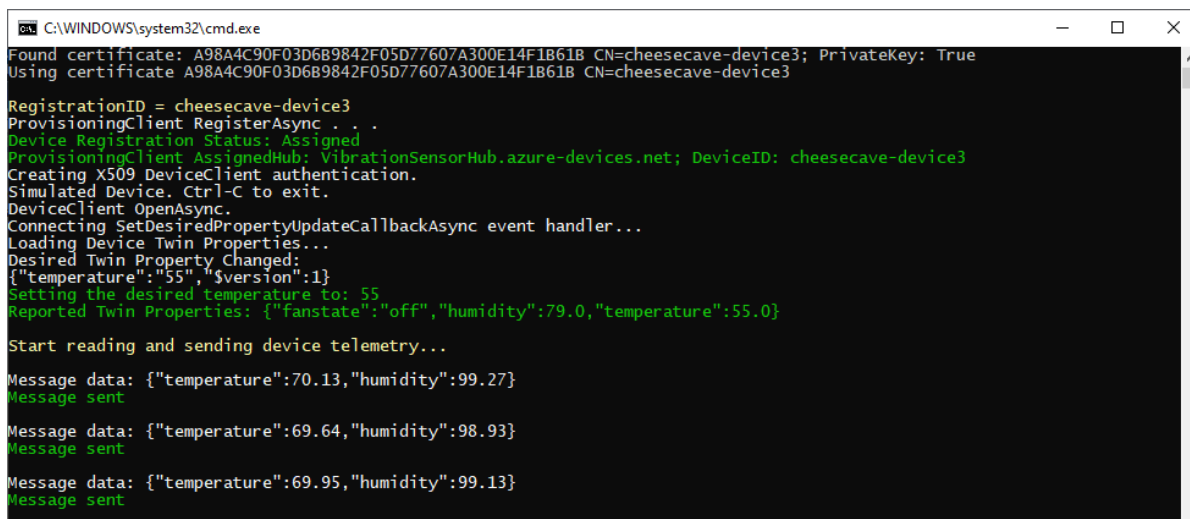


Рисунок 3.3 – Консоль, яка показує, що третій пристрій працює правильно.

3.2. Тестування автоматичної підготовки та призначення в Центр Інтернету речей для декількох пристроїв.

1. Переконаємось, що всі три додатки для пристроїв, як і раніше відправляють дані телеметрії.

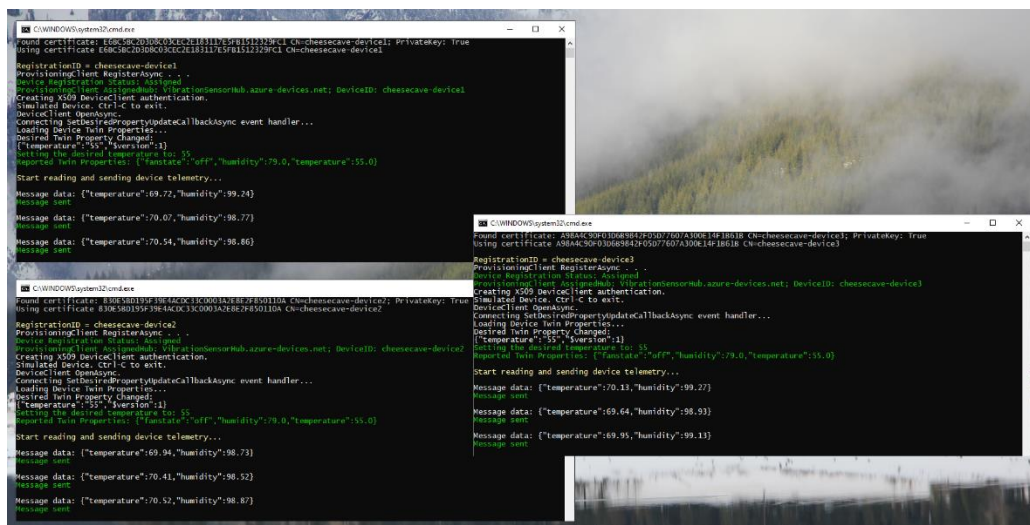


Рисунок 3.4 – Консоль з відображення усіх трьох пристроїв.

2. На порталі «Azure» для центру «Інтернету речей» обираємо «Пристрої IoT» в меню зліва. Переконаємось, що вказані всі три пристрої, а для параметра «Стан» встановлено значення «Вкл».

ИДЕНТИФИКАТОР УСТРОЙСТВА	СОСТОЯНИЕ	ВРЕМЯ ПОСЛЕДНЕГО ОБНОВЛ...	ТИП ПРОВЕРКИ ПОДЛИННОСТИ	ЧИСЛО СООБЩЕНИЙ, ОТПРАВЛЕННЫХ...
cheesecave-device1	Включено	--	Самоподписанный	0
cheesecave-device2	Включено	--	Самоподписанный	0
cheesecave-device3	Включено	--	Самоподписанный	0

Рисунок 3.5 – Знімок екрану, який показує список ідентифікаторів пристроїв, включених в центрі «Інтернету речей».

3. Для впевненості в меню зліва вибираємо «Метрики центру». Вводимо «Підключені пристрої» в полі «Метрика» та встановлюємо для параметра «Агрегація» значення «Максимальна».

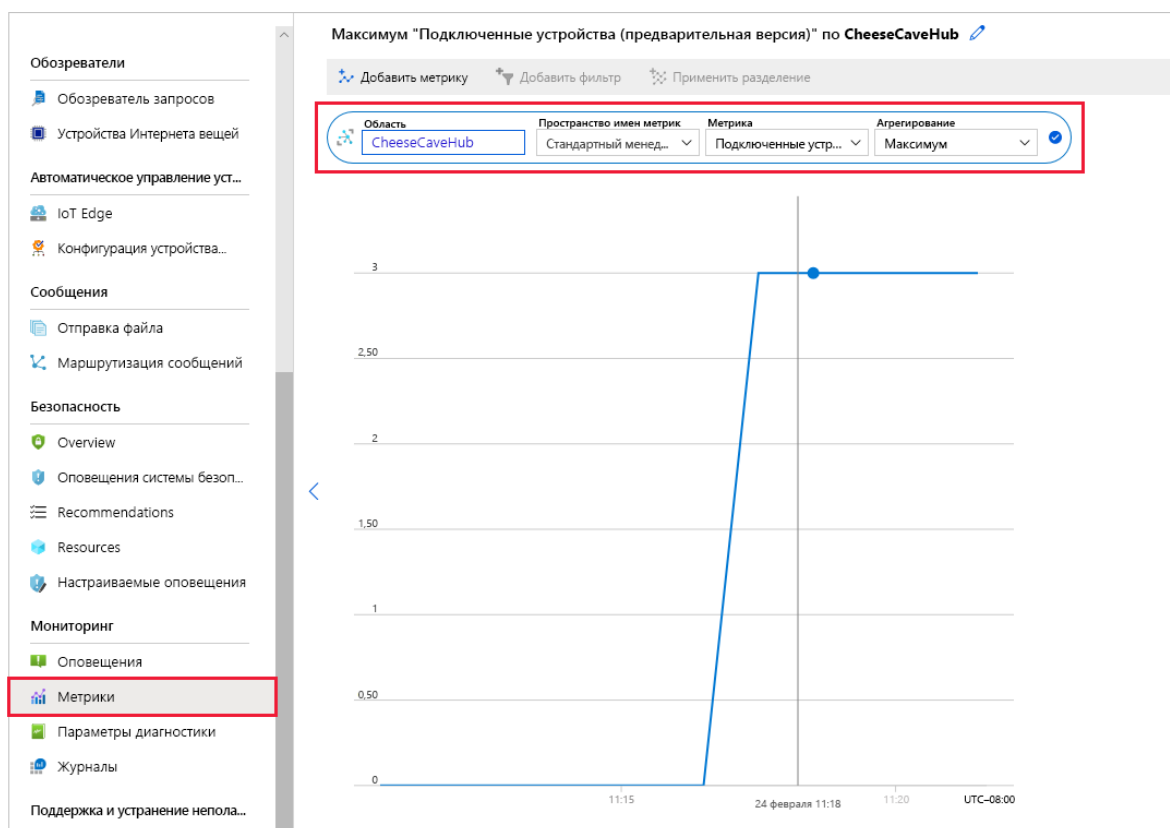


Рисунок 3.6 – Знімок екрану, який показує метрики підключених пристроїв

4. Залишаючись в розділі «Метрики», перевіряємо отримання даних телеметрії, змінивши поле «Метрика» на «Відправлено повідомлень телеметрії» та задавши в полі «Агрегація» значення «Сума».

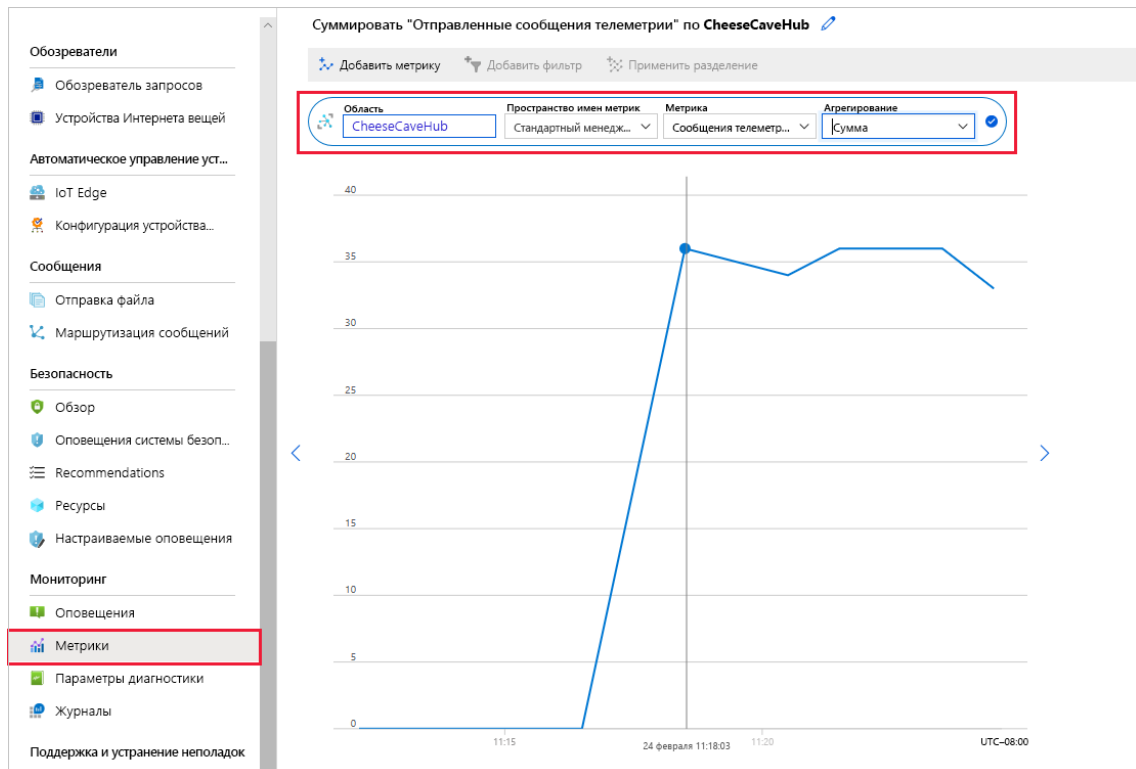


Рисунок 3.7 – Знімок екрану, який показує метрику відправлених повідомлень телеметрії.

Тепер ми переконались, що всі пристрої автоматично підключені до центру через ресурс «DPS». Ми завершили перевірку цього пункту, але ми проведемо ще один тест, щоб перевірити зв'язок в зворотному напрямку.

Тестування обміну даними між хмарою і пристроєм. У цьому тесті потрібно змінити одне з властивостей двійника пристрою на порталі та переконатись, що зміна передається на пристрій. Для цього виконуємо наступні кроки:

1. Вибираємо «Пристрої IoT» для вашого центру, потім вибираємо пристрій «cheesecave-device1». Потім на панелі вибираємо «Двійник пристрою».
2. У кодї, який з'явився відкриваємо розділ «Необхідні властивості».

3. Додаємо помітне необхідне значення температури, наприклад "50,123".
Натискаємо «Зберегти».

Лістинг коду:

"temperature": "50.1234",

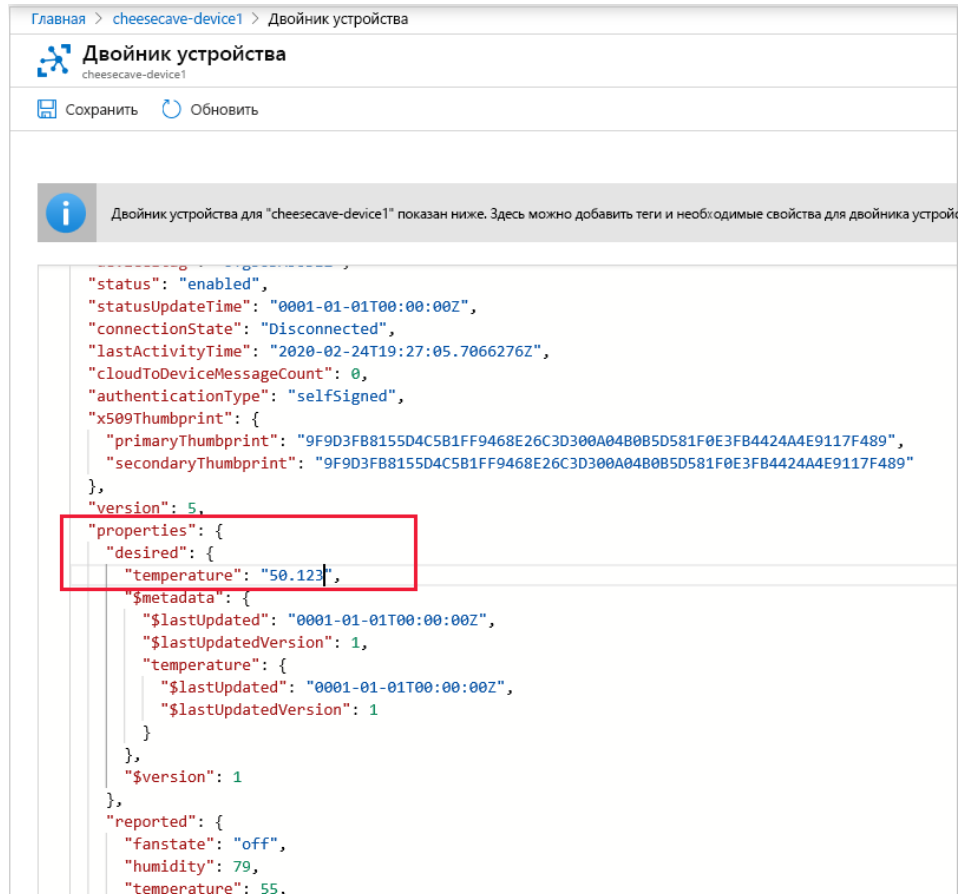


Рисунок 3.8 – Знімок екрану, який показує змінення необхідних властивостей у двійника пристрою

4. Переконаємось в вихідних даних консолі, що ця зміна швидко потрапляє на вибраний пристрій.

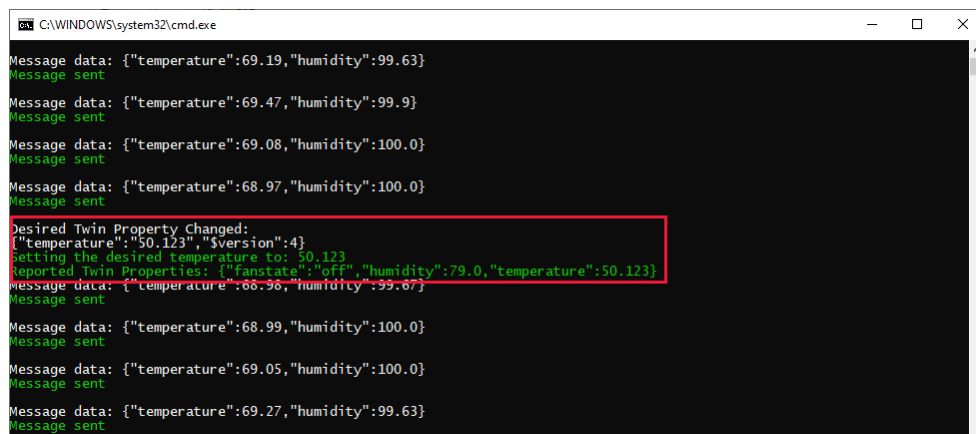


Рисунок 3.9 – Консоль, яка виводить змінення властивостей двійника пристрою

Висновки до 3 розділу

У третьому розділі був проведений підсумок всієї роботи, яку зробили та інформації, яку ми знайшли. А також провели повну перевірку працездатності та ефективності прийняття рішень. А саме ми зробили:

1. Розробили код для пристроїв датчиків.
2. Провели повне тестування автоматичної підготовки та призначення в Центр Інтернету речей для декількох пристроїв. Результати роботи представлені у вигляді скрінів та рисунків.

ВИСНОВКИ

В цій дипломній роботі було розроблено та створено «Проект забезпечення безпеки підключаємих пристроїв IoT до хмарного середовища Azure». Для виконання цієї задачі було розроблений певний план з переліком кроків за якими ми слідували:

Крок 1 - Була проаналізована та обгрунтована актуальність теми.

Крок 2 - Проведений аналіз особливостей хмарного середовища Azure та засобів управління його безпекою. Та виявлено найголовніші переваги цього хмарного середовища.

Крок 3 - Проведений аналіз особливостей стратегій і рішень Інтернету речей Azure IoT.

Крок 4 – Проведений детальний аналіз та наведено приклади еталонної архітектури Microsoft Azure Internet of Things (IoT).

Крок 5 – Проведено аналіз особливостей сертифікації X.509 та де саме використовується цей сертифікат.

Крок 6 - Розробка методики безпечного підключення пристроїв IoT з використанням Служби підготовки пристроїв DPS. Та проаналізовано теоретичні відомості.

Крок 7 - Створення служби підготовки пристроїв в центрі Інтернету речей Azure та кореневого сертифікату.

Крок 8 - Створення декількох кінцевих сертифікатів.

Крок 9 - Розробка коду для пристроїв датчиків.

Крок 10 - Тестування автоматичної підготовки та призначення в Центр Інтернету речей для декількох пристроїв.

У результаті чіткого виконання усіх цих кроків було успішно розроблено, створено та протестовано проект забезпечення безпеки підключаємих IoT до хмарного середовища Azure. Проект функціонує на 100%.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основи Azure, частина 1. Опис основных понятий Azure [Електроний курс] – Режим доступу: <https://docs.microsoft.com/ru-ru/learn/paths/az-900-describe-cloud-concepts/>
2. Основи Azure, частина 2. Опис основных понятий Azure [Електроний курс] – Режим доступу: <https://docs.microsoft.com/ru-ru/learn/paths/az-900-describe-core-azure-services/>
3. Загальні свідчення про Azure IoT [Електроний курс] – Режим доступу: <https://docs.microsoft.com/ru-ru/learn/paths/introduction-to-azure-iot/>
4. Безпечне підключення пристроїв Інтернета речей до хмари [Електроний курс] – Режим доступу: <https://docs.microsoft.com/ru-ru/learn/paths/securely-connect-iot-devices/>
5. Хмарні обчислення [Електроний курс] – Режим доступу: <http://integritysys.com.ua/solutions/pricatecloud-solution/>
6. Microsoft Azure сьогодні [Електроний курс] – Режим доступу: <https://www.itweek.ru/its/article/detail.php?ID=149720>
7. Облачні моделі Microsoft Azure [Електроний курс] – Режим доступу: <https://ucloud.ua/ru/service/microsoft-azure/>
8. Що таке хмарне середовище Microsoft Azure та як його використовувати [Електроний курс] – Режим доступу: <https://guidepc.ru/services/chto-takoe-microsoft-azure-i-kak-ee-mozhno-ispolzovat/>
9. Платформа Microsoft Azure [Електроний курс] – Режим доступу: <https://techexpert.ua/ru/products/platforma-microsoft-azure/>
10. Управління хмарним сервісами Microsoft Azure [Електроний курс] – Режим доступу: <https://itglobal.com/ru-ru/managed-it/cloud-management/microsoft-azure/>

11. Інтернет речей Azure [Електроний курс] – Режим доступу:
<https://azure.microsoft.com/ru-ru/overview/iot/>
12. Еталона архітектура Azure [Електроний курс] – Режим доступу:
<https://docs.microsoft.com/ru-ru/azure/architecture/reference-architectures/iot>
13. Сертифікат Х.509 [Електроний курс] – Режим доступу:
<https://habr.com/ru/post/194664/>
14. Сертифікат Х.509, частина 2 [Електроний курс] – Режим доступу:
<http://kriptografea.narod.ru/x509.html>
15. Служба підготовки пристроїв DPS [Електроний курс] – Режим доступу:
<https://docs.microsoft.com/ru-ru/azure/iot-dps/concepts-roles-operations>

ДОДАТКИ

					КБ.КР.Б – 125 – 20 – ПЗ	Арк.
						65

```
# create certificates directory
mkdir certificates
# navigate to certificates directory
cd certificates
# download helper script files
curl https://raw.githubusercontent.com/Azure/azure-iot-sdk-c/master/tools/CACertificates/certGen.sh --output certGen.sh
curl https://raw.githubusercontent.com/Azure/azure-iot-sdk-c/master/tools/CACertificates/openssl_device_intermediate_ca.cnf --output openssl_device_intermediate_ca.cnf
curl https://raw.githubusercontent.com/Azure/azure-iot-sdk-c/master/tools/CACertificates/openssl_root_ca.cnf --output openssl_root_ca.cnf
# update script permissions so user can read, write, and execute it
chmod 700 certGen.sh
```

```
// Copyright (c) Microsoft. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root for full
license information.
```

```
using Microsoft.Azure.Devices.Client;
using Microsoft.Azure.Devices.Provisioning.Client;
using Microsoft.Azure.Devices.Provisioning.Client.Transport;
using Microsoft.Azure.Devices.Shared;
using System;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using Newtonsoft.Json;
using System.Security.Cryptography.X509Certificates;

namespace X509CertificateSimulatedDevice
{
    class Program
    {
        // Azure Device Provisioning Service (DPS) Global Device Endpoint.
        private const string GlobalDeviceEndpoint = "global.azure-devices-provisioning.net";

        // Azure Device Provisioning Service (DPS) ID Scope.
        private static string dpsIdScope = "<your DPS ID Scope>";
```

```

// Certificate (PFX) File Name.
private static string s_certificateFileName = "<your leaf path>";

// Certificate (PFX) Password. Better to use a Hardware Security Module for
production devices.
private static string s_certificatePassword = "1234";
public static int Main(string[] args)
{
    X509Certificate2 certificate = LoadProvisioningCertificate();
    using (var security = new SecurityProviderX509Certificate(certificate))
    {
        using (var transport = new
ProvisioningTransportHandlerAmqp(TransportFallbackType.TcpOnly))
        {
            ProvisioningDeviceClient provClient =
                ProvisioningDeviceClient.Create(GlobalDeviceEndpoint, dpsIdScope,
security, transport);
            var provisioningDeviceLogic = new
ProvisioningDeviceLogic(provClient, security);
            provisioningDeviceLogic.RunAsync().GetAwaiter().GetResult();
        }
    }
    return 0;
}
private static X509Certificate2 LoadProvisioningCertificate()
{
    var certificateCollection = new X509Certificate2Collection();
    certificateCollection.Import(s_certificateFileName, s_certificatePassword,
X509KeyStorageFlags.UserKeySet);
    X509Certificate2 certificate = null;
    foreach (X509Certificate2 element in certificateCollection)
    {
        Console.WriteLine($"Found certificate: {element?.Thumbprint}
{element?.Subject}; PrivateKey: {element?.HasPrivateKey}");
        if (certificate == null && element.HasPrivateKey)
        {
            certificate = element;
        }
        else
        {
            element.Dispose();
        }
    }
    if (certificate == null)
    {

```

```

        throw new FileNotFoundException($"{s_certificateFileName} did not
contain any certificate with a private key.");
    }
    Console.WriteLine($"Using certificate {certificate.Thumbprint}
{certificate.Subject}");
    return certificate;
}
}

```

// The ProvisioningDeviceLogic class contains the device logic to read from the simulated Device Sensors, and send Device-to-Cloud messages to the Azure IoT Hub. It also contains the code that updates the device with changes to the device twin properties.

```

public class ProvisioningDeviceLogic
{
    readonly ProvisioningDeviceClient _provClient;
    readonly SecurityProvider _security;
    DeviceClient s_deviceClient;
    // Global constants.
    const float ambientTemperature = 70;           // Ambient temperature of a
southern cave, in degrees F.
    const double ambientHumidity = 99;           // Ambient humidity in
relative percentage of air saturation.
    const double desiredTempLimit = 5;           // Acceptable range above or
below the desired temp, in degrees F.
    const double desiredHumidityLimit = 10;       // Acceptable range above or
below the desired humidity, in percentages.
    const int intervalInMilliseconds = 5000;     // Interval at which telemetry
is sent to the cloud.
    enum stateEnum
    {
        off,
        on,
        failed
    }
    // Global variables.
    private static stateEnum fanState = stateEnum.off; // Initial setting
of the fan.
    private static double desiredTemperature = ambientTemperature - 10; // Initial
desired temperature, in degrees F.
    private static double desiredHumidity = ambientHumidity - 20; // Initial
desired humidity in relative percentage of air saturation.
    public ProvisioningDeviceLogic(ProvisioningDeviceClient
provisioningDeviceClient, SecurityProvider security)
    {

```

```

        _provClient = provisioningDeviceClient;
        _security = security;
    }
    private static void colorMessage(string text, ConsoleColor clr)
    {
        Console.ForegroundColor = clr;
        Console.WriteLine(text);
        Console.ResetColor();
    }
    private static void greenMessage(string text)
    {
        colorMessage(text, ConsoleColor.Green);
    }
    private static void redMessage(string text)
    {
        colorMessage(text, ConsoleColor.Red);
    }
    private static void whiteMessage(string text)
    {
        colorMessage(text, ConsoleColor.White);
    }
    public async Task RunAsync()
    {
        colorMessage($"\\nRegistrationID = {_security.GetRegistrationID()}",
ConsoleColor.Yellow);
        // Register the Device with DPS.
        whiteMessage("ProvisioningClient RegisterAsync . . . ");
        DeviceRegistrationResult result = await
        _provClient.RegisterAsync().ConfigureAwait(false);
        if (result.Status == ProvisioningRegistrationStatusType.Assigned)
        {
            greenMessage($"Device Registration Status: {result.Status}");
            greenMessage($"ProvisioningClient AssignedHub: {result.AssignedHub};
DeviceID: {result.DeviceId}");
        }
        else
        {
            redMessage($"Device Registration Status: {result.Status}");
            throw new Exception($"DeviceRegistrationResult.Status is NOT
'Assigned'");
        }
        // Create x509 DeviceClient Authentication.
        whiteMessage("Creating X509 DeviceClient authentication.");
        var auth = new DeviceAuthenticationWithX509Certificate(result.DeviceId,
(_security as SecurityProviderX509).GetAuthenticationCertificate());

```

```

        whiteMessage("Simulated Device. Ctrl-C to exit.");
        using (s_deviceClient = DeviceClient.Create(result.AssignedHub, auth,
TransportType.Amqp))
        {
            // Explicitly open DeviceClient to communicate with Azure IoT Hub.
            whiteMessage("DeviceClient OpenAsync.");
            await s_deviceClient.OpenAsync().ConfigureAwait(false);
            // Setup OnDesiredPropertyChanged Event Handling to receive Desired
Properties changes.
            whiteMessage("Connecting SetDesiredPropertyUpdateCallbackAsync
event handler...");
            await
s_deviceClient.SetDesiredPropertyUpdateCallbackAsync(OnDesiredPropertyChange
d, null).ConfigureAwait(false);
            // Load Device Twin Properties since device is just starting up.
            whiteMessage("Loading Device Twin Properties...");
            var twin = await s_deviceClient.GetTwinAsync().ConfigureAwait(false);
            // Use OnDesiredPropertyChanged event handler to set the loaded Device
Twin properties.
            await OnDesiredPropertyChanged(twin.Properties.Desired, null);
            // Start reading and sending device telemetry.
            colorMessage("\nStart reading and sending device telemetry...\n",
ConsoleColor.Yellow);
            await SendDeviceToCloudMessagesAsync2(s_deviceClient);
            // Explicitly close DeviceClient.
            whiteMessage("DeviceClient CloseAsync.");
            await s_deviceClient.CloseAsync().ConfigureAwait(false);
        }
    }

    private async Task OnDesiredPropertyChanged(TwinCollection
desiredProperties, object userContext)
    {
        whiteMessage("Desired Twin Property Changed:");
        whiteMessage($"{desiredProperties.ToJson()}");
        // Read the desired Twin Properties.
        if (desiredProperties.Contains("fanstate") & fanState != stateEnum.failed)
        {
            string desiredFanState = desiredProperties["fanstate"];
            desiredFanState = desiredFanState.ToLower();
            if (desiredFanState == "on" || desiredFanState == "off")
            {
                fanState = (stateEnum)Enum.Parse(typeof(stateEnum), desiredFanState);
                greenMessage($"Set the fan to: {desiredFanState}");
            }
        }
        else
    }

```

```

        {
            redMessage($"Illegal fan state received: {desiredFanState}");
        }
    }
    if (desiredProperties.Contains("temperature"))
    {
        string desiredTemperatureString = desiredProperties["temperature"];
        try
        {
            desiredTemperature = double.Parse(desiredTemperatureString);
            greenMessage($"Setting the desired temperature to:
{desiredTemperatureString}");
        }
        catch
        {
            redMessage($"Illegal temperature received:
{desiredTemperatureString}");
        }
    }
    if (desiredProperties.Contains("humidity"))
    {
        string desiredHumidityString = desiredProperties["humidity"];
        try
        {
            desiredHumidity = double.Parse(desiredHumidityString);
            greenMessage($"Setting the desired humidity to:
{desiredHumidityString}");
        }
        catch
        {
            redMessage($"Illegal humidity received: {desiredHumidityString}");
        }
    }
    // Report Twin properties.
    var reportedProperties = new TwinCollection();
    reportedProperties["fanstate"] = fanState.ToString();
    reportedProperties["humidity"] = desiredHumidity;
    reportedProperties["temperature"] = desiredTemperature;
    await
s_deviceClient.UpdateReportedPropertiesAsync(reportedProperties).ConfigureAwait
(false);
    greenMessage($"Reported Twin Properties: {reportedProperties.ToJson()}");
}
// Async method to send simulated telemetry.

```

```

private async Task SendDeviceToCloudMessagesAsync2(DeviceClient
deviceClient)
{
    double currentTemperature = ambientTemperature;    // Initial setting of
temperature.
    double currentHumidity = ambientHumidity;          // Initial setting of
humidity.
    Random rand = new Random();
    while (true)
    {
        // Simulate telemetry.
        double deltaTemperature = Math.Sign(desiredTemperature -
currentTemperature);
        double deltaHumidity = Math.Sign(desiredHumidity - currentHumidity);
        if (fanState == stateEnum.on)
        {

            // If the fan is on the temperature and humidity will be nudged towards
the desired values most of the time.
            currentTemperature += (deltaTemperature * rand.NextDouble()) +
rand.NextDouble() - 0.5;
            currentHumidity += (deltaHumidity * rand.NextDouble()) +
rand.NextDouble() - 0.5;
            // Randomly fail the fan.
            if (rand.NextDouble() < 0.01)
            {
                fanState = stateEnum.failed;
                redMessage("Fan has failed");
            }
        }
        else
        {
            // If the fan is off, or has failed, the temperature and humidity will creep
up until they reaches ambient values,
            // thereafter fluctuate randomly.
            if (currentTemperature < ambientTemperature - 1)
            {
                currentTemperature += rand.NextDouble() / 10;
            }
            else
            {
                currentTemperature += rand.NextDouble() - 0.5;
            }
            if (currentHumidity < ambientHumidity - 1)
            {

```

```

        currentHumidity += rand.NextDouble() / 10;
    }
    else
    {
        currentHumidity += rand.NextDouble() - 0.5;
    }
}
// Check: humidity can never exceed 100%.
currentHumidity = Math.Min(100, currentHumidity);
// Create JSON message.
var telemetryDataPoint = new
{
    temperature = Math.Round(currentTemperature, 2),
    humidity = Math.Round(currentHumidity, 2)
};
var messageString = JsonConvert.SerializeObject(telemetryDataPoint);
var message = new Message(Encoding.ASCII.GetBytes(messageString));
// Add custom application properties to the message.
message.Properties.Add("sensorID", "S1");
message.Properties.Add("fanAlert", (fanState == stateEnum.failed) ? "true"
: "false");
// Send temperature or humidity alerts, only if they occur.
if ((currentTemperature > desiredTemperature + desiredTempLimit) ||
(currentTemperature < desiredTemperature - desiredTempLimit))
{
    message.Properties.Add("temperatureAlert", "true");
}
if ((currentHumidity > desiredHumidity + desiredHumidityLimit) ||
(currentHumidity < desiredHumidity - desiredHumidityLimit))
{
    message.Properties.Add("humidityAlert", "true");
}
colorMessage($"Message data: {messageString}", ConsoleColor.White);
// Send the telemetry message
await deviceClient.SendEventAsync(message);
greenMessage("Message sent\n");
await Task.Delay(intervalInMilliseconds);
}
}
}
}

```