Industrial IoT

Case Study – Azure

Dokumentacja projektu

Repozytorium projektu: https://github.com/bgolabczak/IoT--projekt-zaliczenie

AzureDeviceSdkDemo.Device - klasa z funkcjami sterującymi metodami klienta urządzenia IoT.

FunctionAppTriggers kod źródłowy funkcjonalności aplikacji Azure

ProjektZaliczeniowy – main class.

asa-bgolabczak – wyeksportowany kod of Azure Stream Analitycs.

Main class logic (Logika klasy Main class)

1. Aby połączyć się z Klientem Opc, inicjowana jest nowa instancja klasy OpcClient przy użyciu określonego adresu serwera zapisany w obiekcie Resources. Nawiązano połączenie z serwerem klienta OPC za pomocą metody Connect(). Do przeglądania używana jest metoda BrowseNode() obiektu opcClient węzła OpcObjectTypes.ObjectsFolder
2. Zainicjowany obiekt RegistryManager wykorzystuje parametry połączenia przechowywane w obiekcie Resources. Potem użyta jest metoda GetDevicesAsync obiektu RegistryManager w celu uzyskania listy zarejestrowanych urządzeń w IoT Hub, ograniczona do maksymalnej liczby określonej w obiekcie Resources. Następnie kod idzie i przegląda listę urządzeń i sprawdza, czy identyfikator urządzenia zawiera określony ciąg znaków. Jeśli tak, to identyfikator urządzenia dodany do listy identyfikatorów urządzeń.
3. Zainicjowano nowy słownik, który ma klucz typu VirtualDevice i wartość typu MachineData, która jest klasą publiczną używaną do przechowywania danych odczytanych z klienta OPC w czytelny i szybki sposób. Następnie w pętli tworzony jest nowy obiekt DeviceClient przy użyciu parametrów połączenia i bieżącego elementu na liście identyfikatorów urządzeń. Klient urządzenia otwiera się asynchronicznie. Następnie zainicjował nowe urządzenie wirtualne obiekty przy użyciu klienta urządzenia i obiektu opcClient. Programy obsługi są inicjowane dla urządzenia wirtualnego używając właściwości machineId bieżącego elementu na liście machineDataList. Urządzenie wirtualne i jego odpowiedni obiekt machineData jest następnie dodawany do słownika. Wreszcie właściwości iotHubDeviceId obiektu MachineData jest ustawiony na bieżący element na liście DeviceIds
4. Dla każdej pary klucz-wartość w słowniku kod wykonuje następujące czynności:
   Wywołuje funkcję readNode i przekazuje wartość (typu MachineData) oraz opcClient jako argumenty.
   Wywołuje funkcję SetTwinAsync na kluczu (typu VirtualDevice) i przekazuje wartości

deviceErrors i wartości productRate (typu MachineData) jako argumenty.
Jeśli wartość deviceErrors (typu MachineData) jest większa niż 0, oznacza to, że wystąpiły błędy na urządzeniu. Wywoływana jest funkcja reportNewError w celu ustawienia stanu błędu w urządzeniu Twin i wysłania go do IoT Hub.
5. Następnie czwarty punkt jest powtarzany co 5 sekund poprzez odczytanie danych z urządzenia, jeśli jest to urządzenie stan produkcyjny jest aktywny, dane telemetryczne są wysyłane do chmury dodatkowo jako komunikat, jeśli
pojawiają się nowe błędy lub zmienia się ich status, do chmury wysyłany jest także komunikat o błędzie i dane są zaktualizowane w bliźniaczym urządzeniu.

Połączenie do urządzenia (Opc UA server)

Metody połączenia opisane są w powyższym punkcie.

Agent odczytuje dane z urządzenia za pomocą funkcji readNode, która jako argument przyjmuje obiekt klasy
MachineData (opisano) i OpcClient, a następnie zapisuje dane w bieżącym obiekcie MachineData poprzez
odczytywanie ich z niektórych węzłów na urządzeniu.
Aktualizacja danych na urządzeniu odbywa się w klasie OnDesiredPropertyChanged, która służy do śledzenia
informacje w przypadku zmiany pożądanych właściwości urządzenia Twin, aby zmniejszyć tempo produkcji Maszyny
i zgłoś nową właściwość do Device Twin.

Agent Konfiguracji

Konfiguracja pliku ProjektZaliczeniowy znajduje się wProjektZaliczeniowy\Properties\Resources.resx

deviceConnectionString	<deviceConnectionString from IoT Hub>
ownerConnectionString	<ownerConnectionString from IoT Hub >
opcClientServer	opc.tcp://localhost:4840/
iotDevicesMaxCount	100

Wiadomości D2C

Agent odczytuje dane z maszyn co 5 sekund. Konwertuje je na json z danymi telemetrycznymi, które wysyła natychmiast do IoT Hub przy użyciu metody klienta urządzenia SendEventAsync().

Rejestrowane wiadomości wysyłanie są do IoT Hub.

Device Twin

Nowe dane dotyczące szybkości produkcji, błędów urządzenia i daty ich ostatniego wystąpienia są zgłaszane do urządzenia Twin in Metody SetTwinAsync lub UpdateTwinAsync.

```
public async Task UpdateTwinAsync(int deviceError)
        {
            var twin = await client.GetTwinAsync();
            Console.WriteLine($"{DateTime.Now}> Device Twin value was update.");
            Console.WriteLine();

            var reportedProperties = new TwinCollection();
            reportedProperties["DeviceErrors"] = deviceError;
            reportedProperties["LastErrorDate"] = DateTime.Now;

            await client.UpdateReportedPropertiesAsync(reportedProperties);
        }
```

Nowe dane pobrane w żądanej właściwości to dane aktualizacji na maszynie, metoda OnDesiredPropertyChanged to opisane w rozdziale Połączenie z urządzeniem (serwerem Opc UA).

Metody bezpośrednie

Możemy wyróżnić 5 metod bezpośrednich zaimplementowanych w Agencie.

```
    await client.SetMethodHandlerAsync("EmergencyStop", EmergencyStopHandler, userContext);
    await client.SetMethodHandlerAsync("ResetErrorStatus", ResetErrorStatusHandler, userContext);
    await client.SetMethodHandlerAsync("DecreaseProductRate", DecreaseProductRateHandler, userContext);
    await client.SetMethodHandlerAsync("MaintenanceDone", MaintenanceDoneHandler, userContext);
    await client.SetMethodDefaultHandlerAsync(DefaultServiceHandler, userContext);
```

Pierwsze dwie wywołują metodę na maszynie.

Jedna z metod:

```
private async Task<MethodResponse> EmergencyStopHandler(MethodRequest methodRequest, object userContext)
        {
            Console.WriteLine($"\t{DateTime.Now}> METHOD EXECUTED: {methodRequest.Name}");
            string nodeId = (string)userContext;
            object[] result = opcClient.CallMethod(
                    nodeId,
                    nodeId + "/EmergencyStop"
                    );
            return new MethodResponse(0);
        }
```

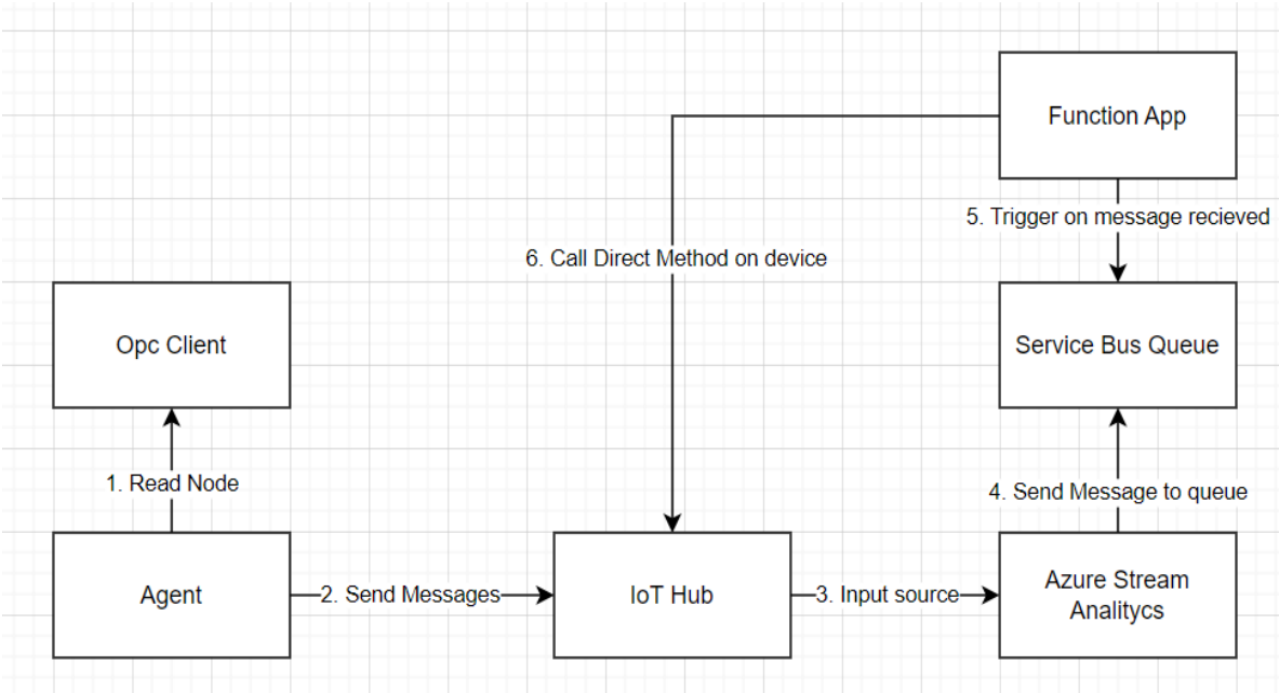
DecreaseProductRate i MaintenanceDone służą do aktualizacji raportowanej właściwości w urządzeniu Twin

```
private async Task<MethodResponse> DecreaseProductRateHandler(MethodRequest methodRequest, object userContext)
        {
            string productionRate = "/ProductionRate";
            string deviceError = "/DeviceError";
            Console.WriteLine($"\t{DateTime.Now}> METHOD EXECUTED: {methodRequest.Name}");
            string nodeId = (string)userContext;
            int rate = (int)opcClient.ReadNode(nodeId + productionRate).Value;
            int error = (int)opcClient.ReadNode(nodeId + deviceError).Value;
            OpcStatus result = opcClient.WriteNode(nodeId + productionRate, rate - 10);
            Console.WriteLine(result.ToString());
            await SetTwinAsync(error, rate - 10);
            return new MethodResponse(0);
        }

private async Task<MethodResponse> MaintenanceDoneHandler(MethodRequest methodRequest, object userContext)
        {
            Console.WriteLine($"\t{DateTime.Now}> METHOD EXECUTED: {methodRequest.Name}");
            var twin = await client.GetTwinAsync();
            var reportedProperties = new TwinCollection();
            reportedProperties["LastMainTenanceDone"] = DateTime.Now;
            await client.UpdateReportedPropertiesAsync(reportedProperties);
            Console.WriteLine($"\n{DateTime.Now}> Device Twin Maintenance Done.");

            return new MethodResponse(0);
        }
```

Logika biznesowa



ASA wysyła komunikat w przypadku spełnienia warunków, co z kolei uruchamia wyzwalacz, który reaguje na nowe wiadomości (message) i natychmiast wywołuje metodę bezpośrednią (direct) na danym urządzeniu.

Wiadomości wysyłane do kolejki zawierają identyfikator urządzenia IoT, na którym chcesz wywołać metodę.