Setting up Kepware - IoT Core Integration

This tutorial show how to setup a KEPServerEX to push streaming data to Google Cloud Platform(GCP) and send control messages from GCP back to KEPServerEX.

Table of Contents

Objectives

- Generate self-signed CA certificate
- Provision IoT device on GCP
- Configure Kepware IoT Gateway with Cloud IoT Core device
- Setup Windows scheduled task to refresh the Json Web Token(JWT)
- Setup IoT Gateway to send simulated metric and verify on Cloud Pub/Sub
- Setup IoT Gateway to receive command message and send command from IoT Core

Costs

This tutorial uses billable components of Google Cloud Platform, including:

- Cloud IoT Core
- Cloud Pub/Sub

Use the Pricing Calculator to generate a cost estimate based on your projected usage.

Before you begin

For this reference guide, you need a GCP project. You can create a new one, or select a project you already created:

1. Select or create a GCP project.

   GO TO THE MANAGE RESOURCES PAGE

2. Enable billing for your project.

   ENABLE BILLING

3. Enable the Cloud IoT Core and Cloud Pub/Sub.

   ENABLE THE APIS

Operating systems requirement

KEPServerEX support following Operating Systems(OS), this tutorial requires you to have a machine with one of the following OS:s installed.

- Windows Server 2019
- Windows Server 2016
- Windows 8 and 10
- Windows 10 IoT Enterprise
- Windows 7 Professional/Enterprise/Ultimate
- Windows Server 2012 and 2012 R2

Powershell

All the programmatic configurations in this tutorial are run from Powershell. It's therefore required you have Powershell version 5.1 or higher installed on your OS.

Install KEPServerEX

- Download JRE8 that matches your OS architecture and install it on your machine
- From KEPServerEX product page download KEPServerEX v6
- Follow the KEPServerEX Install Guide to install KEPServerEX v6 on your windows machine

Enable KEPServerEX Configuration API Service

- Start the KEPServerEX Administration app from Start menu
- From the Administration menu choose **Settings…**
- Choose the **Configuration API Service** tab
- Change the value for Enable to **Yes**
- Change the value for Enable HTTP to **Yes**
- Make sure the value for HTTP Port is set to **57412**
- Set the value of CORS Allowed Origins to *
- Click **Apply** and **Close**

Install Google Cloud SDK

Install Google Cloud SDK on your machine following the SDK installation guide

Generate certificate

Generate a self-signed certificate to identify Kepware as a device in Cloud IoT Core
Following commands are run in PowerShell that has administrative privileges..

1. Generate a self-signed X.509 certificate with a SHA-256 signature and 2048-bit RSA private key.

```
$Cert = New-SelfSignedCertificate -CertStoreLocation
"Cert:\LocalMachine\My" -HashAlgorithm "SHA256" -KeyAlgorithm RSA
-KeyLength 2048  -KeyUsage DigitalSignature -Subject "CN=unused"
```

2. Export the certificate and generate X.509 wrapped public key

```
$WORK_DIR = '<substitute with your working folder to store temporary
files>'

Export-Certificate -Cert $Cert  -Type CERT -FilePath
"$WORK_DIR\rsa_cert.cer"

certutil.exe -encode "$WORK_DIR\rsa_cert.cer" "$WORK_DIR\rsa_cert.pem"
```

Provision resources for IoT connection on GCP

1. Set the GCP project you want to use, substitute the placeholder with your project id:

```
gcloud config set project <your-project-id>
```

2. Create Pub/Sub topic:

```
gcloud pubsub topics create kepware-msg
```

3. Create IoT registry:

```
gcloud iot registries create kepware-devices `
--region europe-west1 `
--event-notification-config=topic=kepware-msg
```

4. Create IoT device:

```
gcloud iot devices create kepware-device `
--region=europe-west1 `
--registry=kepware-devices `
--public-key=path="$WORK_DIR\rsa_cert.pem",type=rsa-x509-pem
```

Create application folder

Create folder storing the configuration files and scripts used in the setup

```
$iotJwtRefresherFolder = "$Env:ProgramData\iotjwtre"
New-Item -ItemType directory -Path  $iotJwtRefresherFolder
&icacls "$iotJwtRefresherFolder" /inheritance:r
&icacls "$iotJwtRefresherFolder" /grant:r "CREATOR OWNER:(OI)(CI)F" /T
&icacls "$iotJwtRefresherFolder" /grant
"BUILTIN\Administrators:(OI)(CI)F" /T
&icacls "$iotJwtRefresherFolder" /grant   "LOCAL SERVICE:(OI)(CI)F" /T
```

Save the configuration file.

```
$configdata = @"
gcp.project=<your-project-id>
cert.path=Cert:\\LocalMachine\\My\\$($Cert.Thumbprint)
"@
$configdata > "$Env:ProgramData\iotjwtre\config.txt"
```

Configure KepServerEx IoT Gateway

1. Declare the function that generates the JWT token:

```
function ConvertTo-Base64-FromString ([parameter(ValueFromPipeline)]
$StringInput) {

Write-Output([Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetByte
s($StringInput)))
}
function Replace-Chars ([parameter(ValueFromPipeline)] $StringInput) {
    Write-Output($StringInput -replace '\+','-' -replace '/','_' -replace
'=')
}
function New-IotCoreJwt ($Algorithm='RS256', $ValidSeconds=7200) {
    $ConfigProps = convertfrom-stringdata (get-content
"$Env:ProgramData\iotjwtre\config.txt" -raw)

    $Header = '{{"alg":"{0}","typ":"JWT"}}' -f $Algorithm
    $Now = [Math]::Floor([decimal](Get-Date (Get-Date).ToUniversalTime()
-UFormat '%s'))
    $Exp = $Now + $ValidSeconds
    $Claim = '{{"aud":"{0}","iat":{1},"exp":{2}}}' -f
$ConfigProps.'gcp.project', $Now, $Exp
    $EncodedHeader = $Header | ConvertTo-Base64-FromString | Replace-Chars
    $EncodedClaim = $Claim | ConvertTo-Base64-FromString | Replace-Chars
    $ToSign = $EncodedHeader + '.' + $EncodedClaim
    $EncodedToSign = [System.Text.Encoding]::UTF8.GetBytes($ToSign)
    $Cert = (Get-ChildItem -Path $ConfigProps.'cert.path')
# If you get an error here, it might
    $Key =
[System.Security.Cryptography.X509Certificates.RSACertificateExtensions]::G
etRSAPrivateKey($Cert)
    $Signature = $Key.SignData($EncodedToSign,
[Security.Cryptography.HashAlgorithmName]::SHA256,
[Security.Cryptography.RSASignaturePadding]::Pkcs1)
    $EncodedSignature = [Convert]::ToBase64String($Signature) |
Replace-Chars
    Write-Output($ToSign + '.' + $EncodedSignature)
}
```

2. Generate a new JWT token

```
$Jwt = New-IotCoreJwt
```

3. Configure IoT Gateway

KEPServerEX IoT Gateway plugin comes with a MQTT client which is configured to connect to MQTT broker of Cloud IoT Core. Configuration is down programmatically through the Kepware Configuration API.

Defines the function that creates the IoT Gateway Agent:

```
function Get-AdminHeader() {
    $EncodedCred =
[System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.GetBytes('Ad
ministrator:'))
    Write-Output(@{ Authorization = "Basic $EncodedCred" })
}

function Create-IoTGatewayAgent ($Jwt, $AgentName='IoTCore',
$Region='europe-west1', $Registry='kepware-devices',
$Device='kepware-device') {
    $ConfigProps = convertfrom-stringdata (get-content
"$Env:ProgramData\iotjwtre\config.txt" -raw)
    $Uri =
'http://127.0.0.1:57412/config/v1/project/_iot_gateway/mqtt_clients'
    $EventsTopic = '/devices/{0}/events' -f $Device
    $ClientId = 'projects/{0}/locations/{1}/registries/{2}/devices/{3}' -f
$ConfigProps.'gcp.project', $Region, $Registry, $Device
    $Body = @{
        "common.ALLTYPES_NAME" = $AgentName;
        "iot_gateway.MQTT_CLIENT_URL" = "ssl://mqtt.googleapis.com:8883";
        "iot_gateway.MQTT_CLIENT_TOPIC" = $EventsTopic;
        "iot_gateway.MQTT_CLIENT_CLIENT_ID" = $ClientId;
        "iot_gateway.MQTT_CLIENT_USERNAME" = "unused";
        "iot_gateway.MQTT_CLIENT_PASSWORD" = $Jwt
    }
    $Payload = ConvertTo-Json $Body
    Invoke-RestMethod -Uri $Uri -Method Post -Headers (Get-AdminHeader)
-Body $Payload
}
```

- Run the function to create the IoT Gateway agent

```
Create-IoTGatewayAgent $Jwt
```

Setup scheduled task to update JWT token

Here you setup a scheduled task that runs a Powershell script to update JWT token in the IoT Gateway configuration. Task is scheduled to run every 1 hour 30 minutes.

1. Create a Powershell script file with content as follows. First replace the placeholder for project id with your project id

```
$scriptContent = @'
function ConvertTo-Base64-FromString ([parameter(ValueFromPipeline)]
$StringInput) {

Write-Output([Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($StringInput)))
}
function Replace-Chars ([parameter(ValueFromPipeline)] $StringInput) {
    Write-Output($StringInput -replace '\+','-' -replace '/','_' -replace
'=')
}
function New-IotCoreJwt ($Algorithm='RS256', $ValidSeconds=7200) {
    $ConfigProps = convertfrom-stringdata (get-content
"$Env:ProgramData\iotjwtre\config.txt" -raw)
    $Header = '{{"alg":"{0}","typ":"JWT"}}' -f $Algorithm
    $Now = [Math]::Floor([decimal](Get-Date (Get-Date).ToUniversalTime()
-UFormat '%s'))
    $Exp = $Now + $ValidSeconds
    $Claim = '{{"aud":"{0}","iat":{1},"exp":{2}}}' -f
$ConfigProps.'gcp.project', $Now, $Exp
    $EncodedHeader = $Header | ConvertTo-Base64-FromString | Replace-Chars
    $EncodedClaim = $Claim | ConvertTo-Base64-FromString | Replace-Chars
    $ToSign = $EncodedHeader + '.' + $EncodedClaim
    $EncodedToSign = [System.Text.Encoding]::UTF8.GetBytes($ToSign)
    $Cert = (Get-ChildItem -Path $ConfigProps.'cert.path')
    $Key =
[System.Security.Cryptography.X509Certificates.RSACertificateExtensions]::GetRSAPrivateKey($Cert)
    $Signature = $Key.SignData($EncodedToSign,
[Security.Cryptography.HashAlgorithmName]::SHA256,
[Security.Cryptography.RSASignaturePadding]::Pkcs1)
    $EncodedSignature = [Convert]::ToBase64String($Signature) |
Replace-Chars
    Write-Output($ToSign + '.' + $EncodedSignature)
}
function Get-AdminHeader() {
    $EncodedCred =
[System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.GetBytes('Ad
ministrator:'))
    Write-Output(@{ Authorization = "Basic $EncodedCred" })
}
function Update-AgentJWT ($AgentName='IoTCore') {
    $Uri =
'http://127.0.0.1:57412/config/v1/project/_iot_gateway/mqtt_clients/{0}' -f
$AgentName
    $AgentObj = Invoke-RestMethod -Uri $Uri -Method Get -Headers
(Get-AdminHeader)
    $Jwt = (New-IotCoreJwt)
    $Body = @{
        "PROJECT_ID" = $AgentObj.PROJECT_ID;
        "iot_gateway.MQTT_CLIENT_USERNAME" = "unused";
        "iot_gateway.MQTT_CLIENT_PASSWORD" = $Jwt
    }
    $Payload = ConvertTo-Json $Body
    Invoke-RestMethod -Uri $Uri -Method Put -Headers (Get-AdminHeader)
-Body $Payload
}
Update-AgentJWT
'@

$scriptContent > "$Env:ProgramData\iotjwtre\updateJWT.ps1"
```

2. Create the scheduled task

```
$taskName = "Kepware IoTGateway JWT Refresh"
$jwtReDir = "$Env:ProgramData\iotjwtre"

$action = New-ScheduledTaskAction -Execute 'PowerShell.exe' `
  -Argument "-ExecutionPolicy Bypass -NoProfile $jwtReDir\updateJWT.ps1" `
  -WorkingDirectory $jwtReDir
$trigger = New-ScheduledTaskTrigger `
  -Once `
  -At (Get-Date) `
  -RepetitionInterval (New-TimeSpan -Hours 1 -Minutes 30) `
  -RepetitionDuration (New-TimeSpan -Days (365 * 20))
```

```
$principal = New-ScheduledTaskPrincipal -UserID "NT AUTHORITY\LOCAL SERVICE"
-LogonType ServiceAccount
Register-ScheduledTask -Action $action -Trigger $trigger -Principal $principal
-TaskName $taskName

$task = Get-ScheduledTask -TaskName "$taskName"
$task.Settings.ExecutionTimeLimit = "PT1M"
Set-ScheduledTask $task
```

Setup metric simulation and verify

1. Add an IoT item to the IoT Gateway agent that emits simulated value.

```
function Create-ServerTag($AgentName='IoTCore') {
    $Uri =
'http://127.0.0.1:57412/config/v1/project/_iot_gateway/mqtt_clients/{0}/iot
_items' -f $AgentName
    $Body = @{
        "iot_gateway.IOT_ITEM_SERVER_TAG" = "Simulation
Examples.Functions.Random1";
        "iot_gateway.IOT_ITEM_USE_SCAN_RATE" = $true;
        "iot_gateway.IOT_ITEM_SCAN_RATE_MS" = 20000;
        "iot_gateway.IOT_ITEM_SEND_EVERY_SCAN" = $true;
        "iot_gateway.IOT_ITEM_ENABLED" = $true
    }
    $Payload = ConvertTo-Json $Body
    Invoke-RestMethod -Uri $Uri -Method Post -Headers (Get-AdminHeader)
-Body $Payload
}
Create-ServerTag
```

2. Verify the generated messages are delivered to Cloud Pub/Sub.

- Create a subscription to the kepware even topic

```
gcloud pubsub subscriptions create verify-kepware-msg `
--topic=kepware-msg
```

- Pull message from Pub/Sub subscription to verify the message delivery

```
gcloud pubsub subscriptions pull verify-kepware-msg --auto-ack --limit 10
```

Command receipt

1. Create new tag under Device1 to receive command messages from cloud.

```
function Create-CommandTag() {
    $Uri =
'http://127.0.0.1:57412/config/v1/project/channels/Channel1/devices/Device1
/tags'
    $Body = @{
        "common.ALLTYPES_NAME" = "IoTCoreCommand";
        "servermain.TAG_ADDRESS" = "S0012"
    }
    $Payload = ConvertTo-Json $Body
    Invoke-RestMethod -Uri $Uri -Method Post -Headers (Get-AdminHeader)
-Body $Payload
}
Create-CommandTag
```

2. Connect the newly created tag to IoT Gateway.

```
function Add-CommandTag-IotGateway($AgentName='IoTCore') {
    $Uri =
'http://127.0.0.1:57412/config/v1/project/_iot_gateway/mqtt_clients/IoTCore
/iot_items' -f $AgentName
    $Body = @{
        "iot_gateway.IOT_ITEM_SERVER_TAG" =
"Channel1.Device1.IoTCoreCommand";
        "iot_gateway.IOT_ITEM_ENABLED" = $true
    }
    $Payload = ConvertTo-Json $Body
    Invoke-RestMethod -Uri $Uri -Method Post -Headers (Get-AdminHeader)
-Body $Payload
}
Add-CommandTag-IotGateway
```

3. Enable the IoT Gateway to listen to command topic from cloud

```
function Enable-IoTGateway-Subscription($AgentName='IoTCore') {
    $Uri =
'http://127.0.0.1:57412/config/v1/project/_iot_gateway/mqtt_clients/{0}' -f
$AgentName
    $AgentObj = Invoke-RestMethod -Uri $Uri -Method Get -Headers
(Get-AdminHeader)
```

```
    $Body = @{
        "PROJECT_ID" = $AgentObj.PROJECT_ID;
        "iot_gateway.MQTT_CLIENT_ENABLE_WRITE_TOPIC" = $true;
        "iot_gateway.MQTT_CLIENT_WRITE_TOPIC" =
"/devices/kepware-device/commands/#"
    }

    $Payload = ConvertTo-Json $Body
    Invoke-RestMethod -Uri $Uri -Method Put -Headers (Get-AdminHeader)
-Body $Payload
}
Enable-IoTGateway-Subscription
```

4. Send command from Cloud IoT Core.

```
gcloud iot devices commands send `
--command-data='[{\"id\": \"Channel1.Device1.IoTCoreCommand\",\"v\":
\"testing\"}]' `
--region=europe-west1 `
--registry=kepware-devices `
--device=kepware-device
```

5. Verify command message is received by the newly created tag.
    a. In KEPServerEX click **Tools** > **Launch OPC Quick Client**
    b. In OPC Quick Client click **Channel1.Device1**
    c. Verify the value of **Channel1.Device1.IoTCoreCommand** is **testing**

Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

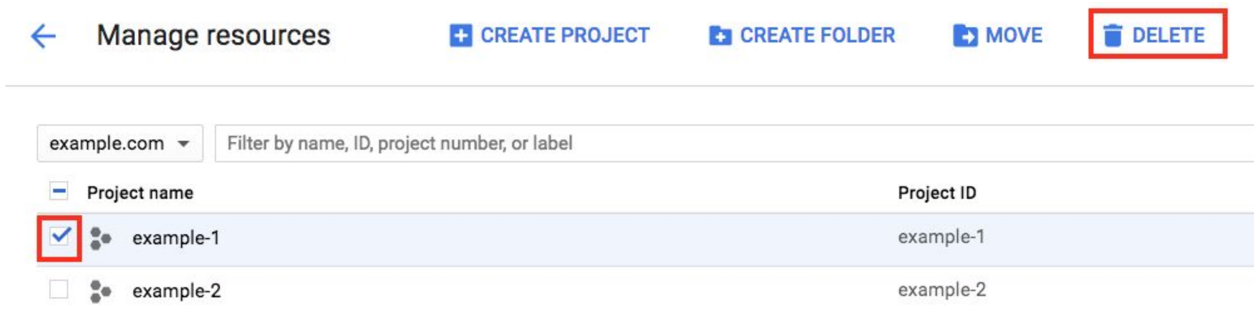
Delete the project

The easiest way to eliminate billing is to delete the project you created for the tutorial.

To delete the project:

1. In the Cloud Platform Console, go to the Projects page.

[GO TO THE PROJECTS PAGE]

2. In the project list, select the project you want to delete and click **Delete**.
3. 

Project name	Project ID
example-1	example-1
example-2	example-2

4. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Delete the individual resources

If you want to keep the project used for this guide, running the following steps to delete the resources you created in this tutorial.

- Delete IoT Core device

```
gcloud iot devices delete kepware-device `
--region=europe-west1 `
--registry=kepware-devices
```

- Delete IoT Core registry

```
gcloud iot registries delete kepware-devices --region=europe-west1
```

- Delete Pub/Sub subscription

```
gcloud pubsub subscriptions delete verify-kepware-msg
```

- Delete Pub/Sub topic

```
gcloud pubsub topics delete kepware-msg
```