FORGEROCK®

Developers Guide

/ ForgeRock® IoT 7.0

Latest update: 7.0.0

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to building applications with the IoT SDK.

Table of Contents

Overview

This guide shows you how to use the IoT SDK to develop client applications and to register them with AM. It also shows you how to build the IoT Gateway.

Quick Start

Develop a client	Build the Gateway
Develop a client application with the IoT SDK.	Build the IoT Gateway for your specific target system.

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

Chapter 1
Develop a Client Application With the IoT SDK

This section shows you how to create a client application for a Thing, named Gopher. The Thing is manually registered in AM and authenticated with a username/password authentication flow. For more information about the IoT SDK API, see the Go package documentation.

- "Develop a ForgeRock IoT Application"
- "Run the Client Application"

Develop a ForgeRock IoT Application

These steps assume that you have installed the required software and cloned the Things GitHub repository:

1. Create a directory structure for your Go project:

   ```
   mkdir -p things/cmd/gopher
   ```

2. Create an empty project file (`main.go`):

   ```
   cd things
   touch cmd/gopher/main.go
   ```

3. Open `main.go` in a text editor, and add the following code:

```
package main

import (
    "github.com/ForgeRock/iot-edge/v7/pkg/builder"
    "github.com/ForgeRock/iot-edge/v7/pkg/callback"
    "log"
    "net/url"
)

func main() {
    amURL, err := url.Parse("http://am.localtest.me:8080/openam")
    if err != nil {
        log.Fatal(err)
    }
    _, err = builder.Thing().
        ConnectTo(amURL).
        InRealm("/").
        WithTree("Example").
        HandleCallbacksWith(
            callback.NameHandler{Name: "Gopher"},
            callback.PasswordHandler{Password: "5tr0ngG3n3r@ted"}).
        Create()
    if err != nil {
        log.Fatal(err)
    }
    log.Println("Gopher successfully authenticated.")
}
```

4. Create a Go module:

   ```
   go mod init example.com/things
   go: creating new go.mod: module example.com/things
   ```

   This step creates a `go.mod` file that specifies your project dependencies and versions.

Run the Client Application

1. Before you can run the application, you need to register an identity for Gopher in AM:

   a. Obtain an admin SSO token from AM:

      ```
      curl \
      --header 'X-OpenAM-Username: amAdmin' \
      --header 'X-OpenAM-Password: changeit' \
      --header 'Content-Type: application/json' \
      --header 'Accept-API-Version: resource=2.0, protocol=1.0' \
      --request POST \
      'http://am.localtest.me:8080/openam/json/authenticate'
      {
       "tokenId": "qGAzvBw20z5...AAA.*",
       "successUrl": "/openam/console",
       "realm": "/"
      }
      ```

   b. Save the `tokenId` returned in this request as a variable, for example:

```
export tokenId=qGAzvBw20z5...AAA.*
echo $tokenId
qGAzvBw20z5...AAA.*
```

c. Register the Gopher application, with the ID `Gopher`:

```
curl \
--header 'Content-Type: application/json' \
--header 'Accept-Api-Version: resource=4.0, protocol=2.1' \
--cookie "iPlanetDirectoryPro=${tokenId}" \
--data '{
    "userPassword": "5tr0ngG3n3r@ted",
    "thingType": "device"
}' \
--request PUT \
"http://am.localtest.me:8080/openam/json/realms/root/users/Gopher"
{
  "_id": "Gopher",
  "_rev": "-1",
  "realm": "/",
  "username": "Gopher",
  "uid": [
    "Gopher"
  ],
  "universalid": [
    "id=Gopher,ou=user,dc=openam,dc=forgerock,dc=org"
  ],
  "objectClass": [
    "iplanet-am-managed-person",
    "inetuser",
    "fr-iot",
    "sunFMSAML2NameIdentifier",
    "inetorgperson",
    "devicePrintProfilesContainer",
    "iplanet-am-user-service",
    "iPlanetPreferences",
    "pushDeviceProfilesContainer",
    "forgerock-am-dashboard-service",
    "organizationalperson",
    "top",
    "kbaInfoContainer",
    "person",
    "sunAMAuthAccountLockout",
    "oathDeviceProfilesContainer",
    "webauthnDeviceProfilesContainer",
    "iplanet-am-auth-configuration-service",
    "deviceProfilesContainer"
  ],
  "dn": [
    "uid=Gopher,ou=people,dc=openam,dc=forgerock,dc=org"
  ],
  "inetUserStatus": [
    "Active"
  ],
  "cn": [
    "Gopher"
  ],
  "sn": [
    "Gopher"
  ],
  "thingType": [
    "device"
  ],
  "createTimestamp": [
    "20200831103235Z"
  ]
}
```

If you sign in to the AM Admin UI and select Identities in the Top Level Realm, you will see the `Gopher` identity in the list.

2. Build an executable for your client application:

```
go build example.com/things/cmd/gopher
go: finding module for package github.com/ForgeRock/iot-edge/v7/pkg/callback
go: finding module for package github.com/ForgeRock/iot-edge/v7/pkg/builder
go: downloading github.com/ForgeRock/iot-edge/v7 v7.0.0
go: downloading github.com/ForgeRock/iot-edge v0.0.0-20200812141306-ee64981fa05f
go: found github.com/ForgeRock/iot-edge/v7/pkg/builder in github.com/ForgeRock/iot-edge/v7 v7.0.0
go: found github.com/ForgeRock/iot-edge/v7/pkg/callback in github.com/ForgeRock/iot-edge/v7 v7.0.0
```

This step builds an executable `gopher` application in the `things` directory.

3. Run the executable to authenticate your application to AM:

```
./gopher
2020/09/01 11:09:49 Gopher successfully authenticated.
```

Chapter 2
Build the ForgeRock IoT Gateway

ForgeRock does not deliver binaries for the IoT Gateway. There are simply too many operating system and architecture combinations to support. The IoT Gateway and the IoT SDK are developed in the Go programming language primarily because it has uncomplicated build tooling and good support for cross-compilation to target systems.

- "Build the IoT Gateway on a Target System"
- "Cross-Compile the IoT Gateway for a Target System"

Build the IoT Gateway on a Target System

These steps assume that you have installed the required software and cloned the Things GitHub repository:

1. On your target system, navigate to the `iot-edge` directory:

   ```
   cd /path/to/iot-edge
   ```

2. Build the IoT Gateway binary:

   ```
   go build -o ./bin/gateway ./cmd/gateway
   ```

   The IoT Gateway binary is now available at `bin/gateway`.

3. (Optional) Run the IoT Gateway with the `--help` flag for available command-line options:

   ```
   ./bin/gateway --help
   Usage:
    gateway [OPTIONS]

    Application Options:
    --url=      AM URL
    --realm=    AM Realm
    --audience= JWT Audience
    --tree=     Authentication tree
    --name=     Gateway name
    --address=  CoAP Address of Gateway
    --key=      The file containing the Gateway's signing key
    --kid=      The Gateway's signing key ID
    --cert=     The file containing the Gateway's certificate
    --timeout=  Timeout for AM communications (default: 5s)
    -d, --debug     Switch on debug

    Help Options:
   ```

```
-h, --help      Show this help message

2020/08/28 10:40:33 Usage:
gateway [OPTIONS]

Application Options:
--url=      AM URL
--realm=    AM Realm
--audience= JWT Audience
--tree=     Authentication tree
--name=     Gateway name
--address=  CoAP Address of Gateway
--key=      The file containing the Gateway's signing key
--kid=      The Gateway's signing key ID
--cert=     The file containing the Gateway's certificate
--timeout=  Timeout for AM communications (default: 5s)
-d, --debug     Switch on debug

Help Options:
-h, --help      Show this help message
```

Cross-Compile the IoT Gateway for a Target System

You can specify a target system with a combination of the `$GOOS` and `$GOARCH` environment variables. This lets you build the IoT Gateway for a variety of operating system and architecture combinations.

For example, to run the IoT Gateway on an `arm` 32-bit processor (for example, a Raspberry Pi 3 running in 32-bit mode), build the IoT Gateway for `linux/arm`, as follows:

1.
```
GOOS=linux GOARCH=arm go build -o ./bin/gateway ./cmd/gateway
```

2. (Optional) For a complete list of environment and cross-compilation targets, see the go Documentation.

   For more build options, see the **go** command environment variables