Thermostat

Microservice-based and Event-driven

J. Kelly Flanagan

Introduction

A traditional thermostat contains a temperature sensor, a heating and air conditioning system controller, and a set point device for selecting the desired temperature. The thermostat takes the desired temperature and the current temperature, and uses this information to control the heating and air conditioning system appropriately. This document describes a microservice-based thermostat designed around an event-driven architecture. This thermostat requires the same functions as a traditional thermostat. but each component will be physically separate and loosely-coupled. Not surprisingly, this system consists of four major components:

1. a heating and air conditioning controller
2. temperature sensor
3. a calendar based temperature set point
4. a thermostat

The modules that require a physical electrical device also have a state equivalent virtual device created in software. In other words, there is a state equivalent software component for the heating and air conditioning controller and the temperature sensor. The state of a temperature sensor is simply its temperature reading. The virtual temperature sensor periodically connects to the physical sensor, obtains the current temperature and stores this in its state. For a reasonable period of time, a reader of the physical or virtual device would acquire the same value. Likewise the virtual heating and air-conditioning controller stores state indicating whether the physical device is idle, heating, cooling, fan running, etc. The temperature set point module and the thermostat function only have software components.

Each software component has an API for configuring its functionality. The modules primarily communicate by raising and responding to events. These modules are implemented in Python 2.7 and are instantiated as Amazon Web Services (AWS) Lambda Functions. Events are raised and responded to using the AWS Simple Notification Service (SNS). Each module stores its state in an AWS DynamoDB database.

While there are four major components, previously described, they are implemented using several smaller microservices. The list below shows the four modules and the submodules or microservices used to implement them.

1. Calendar based temperature set point
   - Desired Temperature service that compares the current desired temperatures for various areas to the corresponding previous desired temperatures stored in its state. For each pair that differ an event is raised indicating the new dissuader temperature. The current desired temperatures are obtained by acquiring events raised by the Current Calendar Events service. T

    - Current Calendar Events service that acquires current calendar events including desired temperatures for various areas. These calendar events are raised as system events.
    - Google Access Token service that acquires an access token from Google to enable the Current Calendar Events service to access associated Google calendars. When an access token is acquired it is raised as an event enabling all interested services to use it. This service must acquire resident authorization to read their calendars. This is accomplished by raising a request event to the Resident Notification service.
    - Resident Notification service stores the residents email address in its state and when a resident notification request is received it sends the resident a system specific email.
2. Heating and Air Conditioning Controller
3. Temperature Sensor
4. Thermostat

Heating and Air Conditioning Controller

Controls
Fan - on / auto
Enable - heat / cool / auto

Heat and cool should never both be on, both off or one or the other is fine.
Heat on / off
Cool on / off

The state should also include how long since the furnace or ac was on. Don't want to cycle too fast.

API

Resource /hvac

Method GET

```
{
        "area1" :
                        {
                                "fan" : "on / auto",
                                "heater" : "on / off",
                                "ac" : "on / off”,
                                “off_time” : time ac or heater was turned off
                        },
        “area2” :       {
                        }
        …
}
```

Calendar Based Temperature Set Point

Microsservice - resident_notification

This microservice sends a notification to the resident of the home.

API

Resource /resident_notification

Method **GET** Get the notification settings including the email address and SMS numbers where the owner wishes to be notified and the indicators describing how the owner chooses to be notified.

Body No body

Return Format**:**

```
{
        “sms_number” : string - number where SMS messages are to be sent,
        “email_address” : string - email address where notifications are sent,
        “sms_notify” : boolean - true / false to enable or disable SMS,
        “email_notify” : boolean - true / false to enable or disable email
}
```

Example:

```
{
        “sms_number” : “1234567890”,
        “email_address” : “test@example.com”,
        “sms_notify” : false,
        “email_notify” : true
}
```

Method **PUT** Set the notification settings including the email address and SMS numbers where the resident wishes to be notified and the indicators describing how the resident chooses to be notified. **SMS notification doesn’t currently work; enabling it results in a 404 error.**

Body Format**:**

```
{
        “sms_number” : string - number, E.164 format, where SMS are sent,
        “email_address” : string - email address where notifications are sent,
        “sms_notify” : boolean - true / false to enable or disable SMS,
        “email_notify” : boolean - true / false to enable or disable email
}
```

Example:

```
{
        “sms_number” : “+11234567890”,
        “email_address” : “test@example.com”,
        “sms_notify” : false,
        “email_notify” : true
}
```

Returns Nothing returned

Method **POST** Send a notification to the resident.

Body Format:

```
{
        "body" : string - body of email message,
        "subject" : string - subject of email message,
        “from_address” : string - email address of sender
}
```

Example:

```
{
        "body" : "This is my first notification via the API.",
        "subject" : "test notification",
        "from_address" : "devices@flanagan.io"
}
```

Returns Nothing returned

Events

Listen **ResidentNotificationRequest**
arn:aws:sns:us-west-2:356335180012:resident_notification_request

event[‘Records'][0]['Sns']['Subject'] contains,
“ResidentNotificationRequest”

event[‘Records'][0]['Sns']['Message'] contains,

```
“{
        “body" : "This is my first email notification",
        "subject" : "Test Subject",
        "from_address" : ”devices@flanagan.io”
}”
```

json.loads() returns the object

Raise None

Microsservice - google_access_token

This microservice responds to GoogleAccessTokenRequest events by raising a GoogleAccessToken event with a body holding a google access token. The token is either returned from the service's state if still valid, refreshed if necessary, and through the token acquisition process if needed. This service also reacts to the 1_Minute event when a token authorization request is pending. If fatal errors occur a GoogleAccessTokenFailure event is raised.

API

Resource /google_access_token

Method **GET** Get the configuration information for the google_access_token microservice.

Body No body

Return Format:

```
{
        “google_client_id” : string - provided by Google,
        “google_device_host” : string - Google host providing user code and url,
        “google_device_path” : string - path on google_device_host,
        “google_scopes” : string - scopes defining what access is desired,
        “google_token_host” : string - Google host where tokens are acquired,
        “google_token_path” : string - path on google_token_host,
        “google_token_test_path” : string - path on token host to test token
}
```

Example:

```
{
        “google_client_id” : “349653-6ul1fn3lttpffj.apps.googleusercontent.com”,
        “google_device_host” : “accounts.google.com",
        “google_device_path” : “/o/oauth2/device/code",
        “google_scopes” : “https://www.googleapis.com/auth/calendar.readonly",
        “google_token_host” : “www.googleapis.com",
        “google_token_path” : “/oauth2/v4/token",
        “google_token_test_path” : “/oauth2/v1/tokeninfo"
}
```

Method **PUT** Set the configurable elements of the google_access_token service**.**

Body Format:

```
{
        “google_client_id” : string - provided by Google,
        “google_client_secret” : string - provided by Google,
        “google_device_host” : string - Google host providing user code and url,
        “google_device_path” : string - path on google_device_host,
        “google_scopes” : string - scopes defining what access is desired,
        “google_token_host” : string - Google host where tokens are acquired,
        “google_token_path” : string - path on google_token_host,
        “google_token_test_path” : string - path on token host to test token
}
```

Example:

```
{
        “google_client_id” : “34976543-6u1efnttpffj.apps.googleusercontent.com”,
        “google_client_secret” : “*******************”,
        “google_device_host” : “accounts.google.com",
        “google_device_path” : “/o/oauth2/device/code",
        “google_scopes” : “https://www.googleapis.com/auth/calendar.readonly",
        “google_token_host” : “www.googleapis.com",
        “google_token_path” : “/oauth2/v4/token",
        “google_token_test_path” : “/oauth2/v1/tokeninfo"
}
```

Returns Nothing returned

Events

Listen **1_Minute**

arn:aws:sns:us-west-2:356335180012:1_Minute

event[‘Records'][0]['Sns']['Subject'] contains,
“1_Minute”

event[‘Records'][0]['Sns']['Message'] contains,

```
“{
        “minutes" : “minutes since beginning of our time",
}”
```

json.loads() returns the object

GoogleAccessTokenRequest

arn:aws:sns:us-west-2:356335180012:google_access_token_request

event[‘Records'][0]['Sns']['Subject'] contains,
“GoogleAccessTokenRequest”

event[‘Records'][0]['Sns']['Message'] contains,

```
“{
        "subject" : “GoogleAccessTokenRequest”,
        “requester” : “function making original request causing this event"
}”
```

json.loads() returns the object

Raise **GoogleAccessTokenFailure**
arn:aws:sns:us-west-2:356335180012:google_access_token_failure

event[‘Records'][0]['Sns']['Subject'] contains,
“GoogleAccessTokenFailure”

event[‘Records'][0]['Sns']['Message'] contains,
```
“{
        "subject" : “GoogleAccessTokenFailure”,
        “requester” : “function making original request causing this event"
}”
```
json.loads() returns the object

ResidentNotificationRequest
arn:aws:sns:us-west-2:356335180012:resident_notification_request

event[‘Records'][0]['Sns']['Subject'] contains,
“ResidentNotificationRequest”

event[‘Records'][0]['Sns']['Message'] contains,
```
“{
        “body" : "This is my first email notification",
        "subject" : "Test Subject",
        "from_address" : ”devices@flanagan.io”
}”
```
json.loads() returns the object

GoogleAccessToken
arn:aws:sns:us-west-2:356335180012:google_access_token

event[‘Records'][0]['Sns']['Subject'] contains,
“GoogleAccessToken”

event[‘Records'][0]['Sns']['Message'] contains
```
“{
        "subject" : "GoogleAccessToken",
        “access_token” : “google access token”,
        “requester” : “function making original request causing this event"
}”
```
json.loads() returns the object

Microsservice - current_calendar_events

This microservice responds to the CurrentCalendarEventsRequest event and returns a JSON object describing all of the current events on all calendars accessible to the service. Current events are defined as the events that have start times before the current time and have end times after the current time.

API

Resource	/current_calendar_events
Method	**GET** Get the configuration information for the current_calendar_events microservice.
Body	No body

Return Format:

```
{
        “google_calendar_host” : string - Google host for calendar API,
        “google_calendar_path” : string - path on google_calendar_host
}
```

Example:

```
{
        “google_calendar_host” : “www.googleapis.com”,
        “google_calendar_path” : /calendar/v3”

}
```

Method **PUT** Set the configurable elements of the current_calendar_events service**.**

Body Format

```
{
        “google_calendar_host” : string - Google host for calendar API,
        “google_calendar_path” : string - path on google_calendar_host
}
```

Example:

```
{
        “google_calendar_host” : “www.googleapis.com”,
        “google_calendar_path” : /calendar/v3”
}
```

Returns Nothing returned

Events

Listen **GoogleAccessToken**
arn:aws:sns:us-west-2:356335180012:google_access_token

event[‘Records'][0]['Sns']['Subject'] contains,
“GoogleAccessToken”

event[‘Records'][0]['Sns']['Message'] contains
“{
    "subject" : "GoogleAccessToken",
    “access_token” : “google access token”,
    “requester” : “function making original request causing this event"
}”
json.loads() returns the object

CurrentCalendarEventsRequest
arn:aws:sns:us-west-2:356335180012:current_calendar_events_request

event[‘Records'][0]['Sns']['Subject'] contains,
“CurrentCalendarEventsRequest”

event[‘Records'][0]['Sns']['Message'] contains
“{
    "subject" : “CurrentCalendarEventsRequest”
    “requester” : “function making original request causing this event"
}”
json.loads() returns the object

Raise **GoogleAccessTokenRequest**
arn:aws:sns:us-west-2:356335180012:google_access_token_request

event[‘Records'][0]['Sns']['Subject'] contains,
“GoogleAccessTokenRequest”

event[‘Records'][0]['Sns']['Message'] contains,
“{
    "subject" : “GoogleAccessTokenRequest”,
    “requester” : “function making original request causing this event"
}”
json.loads() returns the object

CurrentCalendarEvents
arn:aws:sns:us-west-2:356335180012:current_calendar_events

event[‘Records'][0]['Sns']['Subject'] contains,
“CurrentCalendarEvents”

event[‘Records'][0]['Sns']['Message'] contains
“{
    "subject" : "CurrentCalendarEvents",
    "events" : events - CurrentCalendarEventsObject,
    “requester” : “function making original request causing this event"
}”

json.loads() returns the object

CurrentCalendarEventsObject - at event[‘Records’][0][‘Sns’][‘Message’]['events']

```
{
        “area1” :       [
                                { “event_type” : value },
                                { “event_type” : value },
                                …
                        ],
        “area2” :       [
                                { “event_type” : value },
                                { “event_type” : value },
                                …
                        ]
}
```

Microsservice - desired_temperature

This microservice periodically acquires the desired temperature of a location and if it differs from the previous desired temperature it raises an event to broadcast the new desired value.

API

Resource /desired_temperature

Method **GET** Returns a JSON object indicating the areas of interest and their desired temperatures.

Body No body

Return Format:

```
{
        “area” : integer - desired temperature,
        …
}
```

Example:

```
{
        “NWUpstairsBedroom” : 70,
        “NEUpstairsBedroom” : 72,
        “FamilyRoom” : 68
}
```

Method **PUT** Set the desired temperatures for existing areas. Additional areas in the body are ignored.

Body Format:

```
{
        “area” : integer - desired temperature,
        …
}
```

Example:

```
{
        “NWUpstairsBedroom” : 70,
        “NEUpstairsBedroom” : 72,
        “FamilyRoom” : 68
}
```

Returns Nothing returned

Method **POST** Set the desired temperatures for existing and new areas.

Body Format:

```
{
        “area” : integer - desired temperature,
        …
}
```

Example:

```
{
        “NWUpstairsBedroom” : 70,
        “NEUpstairsBedroom” : 72,
        “FamilyRoom” : 68,
        “NewRoomToAdd”, 74
}
```

Returns Nothing returned

Method **DELETE** Delete area(s) and associated values. If the areas don’t exist they are ignored. The values added in the JSON are ignored as well.

Body Format:

```
{
        “area” : integer - desired temperature,
        …
}
```

Example:

```
{
```

```
            “NWUpstairsBedroom” : 70,
            “NEUpstairsBedroom” : 72,
            “FamilyRoom” : 68,
            “NewRoomToAdd”, 74
}
```

Returns Nothing returned

Events

Listen

5_Minute
arn:aws:sns:us-west-2:356335180012:5_Minute

event[‘Records'][0]['Sns']['Subject'] contains,
“5_Minute”

event[‘Records'][0]['Sns']['Message'] contains,

```
“{
        “minutes" : “minutes since beginning of our time",
}”
```

json.loads() returns the object

CurrentCalendarEvents
arn:aws:sns:us-west-2:356335180012:current_calendar_events

event[‘Records'][0]['Sns']['Subject'] contains,
“CurrentCalendarEvents”

event[‘Records'][0]['Sns']['Message'] contains

```
“{
        "subject" : "CurrentCalendarEvents",
        "events" : events - CurrentCalendarEventsObject,
        “requester” : “function making original request causing this event"
}”
```

json.loads() returns the object

CurrentCalendarEventsObject - at event[‘Records’][0][‘Sns’][‘Message’]['events']

```
{
        “calendar1” :   [
                                { “event_type” : value },
                                { “event_type” : value },
                                …
                        ],
        “calendar2” :   [
                                { “event_type” : value },
                                { “event_type” : value },
                                …
                        ]
}
```

Raise **CurrentCalendarEventsRequest**
arn:aws:sns:us-west-2:356335180012:current_calendar_events_request

event[‘Records'][0]['Sns']['Subject'] contains,
“CurrentCalendarEventsRequest”

event[‘Records'][0]['Sns']['Message'] contains
```
“{
        "subject" : “CurrentCalendarEventsRequest”
        “requester” : “function making original request causing this event"
}”
```
json.loads() returns the object

DesiredTemperature
arn:aws:sns:us-west-2:356335180012:desired_temperature

event[‘Records'][0]['Sns']['Subject'] contains,
“DesiredTemperature”

event[‘Records'][0]['Sns']['Message'] contains,
```
“{
        "subject" : "DesiredTemperature",
        "area" : “area of interest”,
        "temperature" : temperature - integer,
        “requester” : “function making original request causing this event”
}”
```
json.loads() returns the object

Microsservice 2 - Implement API to adjust desired temperature behavior
API:

- Resource: /desired-temperature/level/room
  - PUT: update the dynamoDB database with source of desired temperature data, query interval, and desired temperature.
  - GET: acquire the source of desired temperature data, query interval, and desired temperature.
- Representation
```
    {
        “query-interval” : integer defining query interval, 0 results in no queries,
        “query-location” : URL where query is to be made,
        “desired-temperature” : integer defining desired temperature
    }
```

If query-interval is 0 raise a desired-temperature event if a temperature is PUT.
If query-interval is greater than 0 raise an event if the query results in a temperature different from the previous desired temperature.
If the query-interval is greater than 0 and the query could not be made log the error.

Microservice that raises an event when the physical temperature in the space changes from previous measurement, space-temperature event. This is the virtual instantiations of the associated physical temperature sensor.

Microservice that collects a change-of-desired-temperature event and a space-temperature event and set the desired HVAC controller state. This is the virtual instantiations of the physical HVAC controller.

Things to try and do

ll microservices must have their own API and store and not share data with database.

To send events the micro service uses an existing or creates an appropriate SNS topic and sends events to it. If anyone cares their queues will subscribe to the topic.

Accomplishments / journal

Created API Gateway definition of myHome API. First resource created was desiredTemperature with a sub resource of {level} which is a path parameter that represents the level in the home. The lowest level is 0 and we work up from there. It has a sub resource of {room} which is simply a string representing a room on the level.

This API had a mapping function added to it that looks like this,

```
{
        "level" : "$input.params('level')",
        "room" : "$input.params('room')"
}
```

This enables the path parameters to be supplied to the Lambda function as inputs.

A custom domain name was created for this API. It is

api.kelly.flanagan.io

A CNAME entry was added at my domain pointing api.kelly.flanagan.io to d90ygq2ix5oxu.cloudfront.net. A path was added in the API Gateway to point api.kelly.flanagan.io/myHome to this specific API.

This API was connected to a lambda function written in Python

By defining roles and policies in IAM you don't have to store or access credentials in lambda or other was services. Pretty cool.

When done review the rights granted to each lambda function to see what can be removed.

AWS Specific Implementation Details

Body mapping template added in “integration Request” in the API Gateway. It is of type application/json and has the following implementation,

```
{
        "resource_path" : "$context.resourcePath",
        "http_method" : "$context.httpMethod"
}
```

Interesting things learned:

sns.publish() advertises that it accepts a parameter PhoneNumber, but actually does not. I can't figure out how to send a SMS message via sns. Stackoverflow question asked.

sns.subscribe() subscribes to a sns topic, but messages to the topic do not invoke the lambda function.

The following example code in a lambda function illustrates a simple but easily forgotten fact.

```
Main_function:
   If event X:
      D = func()
   If event Y:
      E = func(D)
```

This errors claiming that D was used before being declared or assigned. It is a fact because each occurs on a different invocation of the main lambda function. While D is assigned during one invocation it is not in the other.

private data that should not be shared

```
{
  "google_device_host": "accounts.google.com",
  "google_token_test_path": "/oauth2/v1/tokeninfo",
  "google_token_path": "/oauth2/v4/token",
  "google_device_path": "/o/oauth2/device/code",
  "google_token_host": "www.googleapis.com",
  "google_scopes": "https://www.googleapis.com/auth/calendar.readonly",
  "google_client_id":
"76914165186-6uilf1e5r7soi5hot80rk3fn3lttpffj.apps.googleusercontent.com"
}
```

Principles:

- DynamoDB tables for lambda function state must be named with the same name as the lambda function.
- ARNs that the lambda function needs must be stored in their DynamoDB state table.

Assume the existence and ownership of an Amazon AWS account, a credentials folder on your local system with AWS credentials, a domain of ones own with a subdomain of api, and that the domain has server certificates. Create a program that does the following:

1. Creates a top level API resource with a configurable name. This is done with swagger document and boto3 create_rest_api
    1. The name is configured in the config files mkSpace.cfg and API.json
    2. Have to add integration templates and lambda arns to get the API tied to the correct lambda function.
2. Installs a lambda function that implements the POST method. Work still remains here to get this working.
    1. Makefile commits files to github.
    2. Config file points to mySpace zip file on github
    3. Downloads the github zip file and installs it on AWS lambda.
    4. Must implement code in mySpace.py
3. The POST method will know where to go get zip files implementing other lambda function and other configuration stuff. Modules will be stored at github

Next steps:

4. Separate the creation of role and attaching a policy. That will make deleting policy and roles clearer.
5. Check returns from all boto3 requests so we capture exceptions and errors.
6. Implement add templates to API to fix build time specifics like lambda ARNs.
7. Determine if there are other dynamic aspects of API create that have to happen during run time / build
8. Implement POST to add to the API and add lambda functions from GitHub
    1. Must add dynamoDB tables, SNS topics and subscriptions
    2. As an example clock_tick service:
        1. Create SNS topics 1_Minute, 5_Minute, etc.
        2. Get ARNs for the SNS topics
        3. Create clock_tick dynamoDB table and store SNS ARNs in it
        4. Download clock_tick zip file from github
        5. Install it as a lambda
        6. Grant it a role that can publish to SNS
    3. Next example google_access_token:
        1. Create SNS topics for google_access_token and google_access_token_request and capture their ARNs
        2. Get 1_Minute SNS ARN
        3. Place ARNS for the three above topics in dynamoDB state table
        4. Download zip file and instantiated function
        5. Grant appropriate role
        6. Add google_access_token API to mySpace
        7. Add uri templates give the lambda arns

9. Implement GET to list available services
10. Implement DELETE to remove resources and services

Top level resource needs to do the following
GET returns a list of installed services
PUT updates an existing service
POST installs a new service
DELETE deletes a service

GET might for example return:
desired_temperature
clock_tick
google_access_token
current_calendar_events
heating_cooling
resident_notification

PUT would reinstall or update the services including their functions that implement their methods

POST installs the service and all code that implements them

DELETE deletes a service and all underlying resources.

The creation of a myHome API includes a set of services that are home wide:
current_calendar_events
resident_notification
clock_tick
google_access_token

These could be zip files stored on github or other website. Then a browser app or a shell script can be run the AWS CLI and upload the lambda, created the associated DynamoDBs, and the API in the API Gateway. This installation could also install a microservice installer that responds to POSTs to the myHome resource to install other services.

A POST to myHome with a payload is essentially a constructor for a service such as desired_temperature. This installs a lambda function that responds to POST requests to define areas for which the temperature is desired, creates the calendar, database, lambda function and API.

A post to myHome could take a payload that describes a service to add.
desired_temperature
heating_and_cooling