COMMAND

Challenge

IoT solutions are expected to interact with devices in such a way that the solution, or people using the solution, may reliably ask devices to perform an action. Furthermore, this interaction must occur across intermittent networks, often using devices with limited resources.

Solution

IoT solutions use the Command pattern to ask devices to perform an action and ensure reliable interactions by leveraging a simple concept: no requested action is deemed successful unless it is acknowledged as successful.

The Command pattern shown in the following diagram can deliver this functionality.

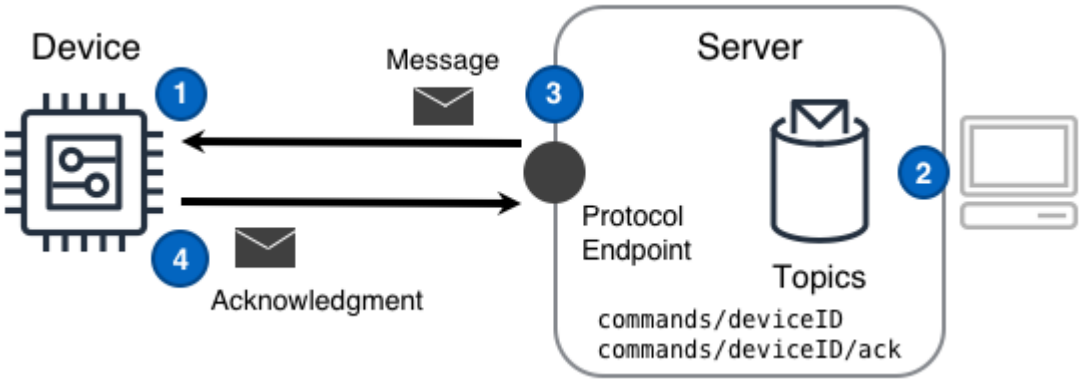

Diagram Steps

1. A device configures itself to communicate with a protocol endpoint so that Command messages can be sent and received.

2. A component of the solution publishes a Command message targeted at one or more devices.
3. The server uses the protocol endpoint to send the Command message to each previously configured device.
4. Upon completion of the Command's requested action, the device publishes a command completion message to the server via the protocol endpoint.

Considerations

It is important to note that the Command pattern is not "telemetry in reverse". Instead the Command pattern addresses the challenge inherent for a solution that needs to reliably trigger actions on a device operating in a remote location.

When implementing this pattern, consider the following questions:

Have you first considered the Device State Replica?

Since the execution of commands actually results in the change of state in a device, the Device State Replica (DSR) pattern is the preferred method of executing commands in an IoT solution. In situations where DSR isn't suitable or exceeds some implementation limits, then consider this Command pattern and a custom implementation of control.

How can the solution track each device's command progress?

Each command should have a solution unique type and each command message should contain a globally unique message ID. The command message's ID allows the solution to track the status of distinct commands and the command type enables the diagnoses any potential issues across categories of commands over time. The messages should be idempotent and not allow for being missed or duplicated without knowledge of the device *and* requestor.

Do some commands in the solution run significantly longer than the norm?

When some commands run longer than the norm, a simple `SUCCESS` or `FAILURE` command completion message will not suffice. Instead the solution should leverage at least three command states: `SUCCESS`, `FAILURE`, and `RUNNING`. `RUNNING` should be returned by the device on an expected interval until the command's completion. By using a `RUNNING` state reported on an expected interval, a solution can determine when a long-running command actually fails quietly.

Does a specific type of command require human authorization?

When a command in a solution requires human approval before a device should take action, a human-workflow component should be added to the solution. This component would intercept commands of particular types and queue them up for human approval before actually sending them onward to a device.

Does a type of command ever need to be rolled back to a previous state?

If a solution has some commands which may need to be rolled back, it is almost always easier to manage that rollback from the solution itself instead of expecting each device to understand and remember the rollback considerations. For example, a device is sent a command to move an actuator from a current, reported position of `0`° to a position of `45`°. The Device performs this command successfully. At a later moment in time the solution requires that the device go back to the previous state, the previous state is often easier to track in the solution itself versus expecting every device to track its former state(s). The rollback of this situation would be performed by the solution sending a command for the device to change position to `0`°.
In the case where rollbacks are necessary even when there is no connectivity to the server, the solution can leverage a gateway to record the former states of devices and to perform rollbacks based upon those values.

Examples

Simple request/response over reliable transport

A component issues a request to a device to actuate a motor, using a quality of service to guarantee delivery MQTT --> QoS = 1

Component sends a command to a target device

The component sends the command request message to the `commands/deviceID` topic to which the device is subscribed:

```
{
  "cmd": "MOTOR_1_ON",
  "tid": "CAFED00D",
  "status": "REQUEST"
}
```

The component also tracks the message's transaction ID of `CAFED00D`, as issued and outstanding for the device.

Device processes message

The device receives the message from topic `commands/deviceID`, activates `motor 1`, and responds with an acknowledgement on the topic `commands/deviceID/ack` to which the component is subscribed. The component receives the following acknowledgment after a period of time:

```
{
  "tid": "CAFED00D",
  "status": "SUCCESS"
}
```

Device and component complete command (transaction) process

The device no longer tracks the command request. The component maps the `SUCCESS` value to the transaction ID of `CAFED00D` and removes the transaction from the list of outstanding requests, signifying the command has completed. A result of `FAILURE` might indicate a physical device problem to be investigated.

Transaction to offline or unavailable device

A component issues a request to a device to actuate a motor, but the device is offline

Component sends command to unavailable device

The component sends the command request message to the `commands/deviceID` topic to which the device is subscribed:

```
{
  "cmd": "MOTOR_1_ON",
  "tid": "CAFED00D",
  "status": "REQUEST"
}
```

The component also tracks the message transaction ID of `CAFED00D`, as issued and outstanding for the device. **The device is offline and does not receive the message.**

Timeout and reissue command

After a set period of time, the component will resend the command request message on a linear or back-off time period *with the same transaction ID*, and tracks the retry status. After a set amount of retries, the component will determine the device did not received the command, or was unable to reply, and take appropriate action.