dojot Documentation

Release 0.4.3

Matheus Magalhaes

set. 04, 2020

Conteúdo:

Esta é a documentação de alto nível para a plataforma dojot IoT desenvolvida pelo CPqD. Esta plataforma visa proporcionar aos desenvolvedores de aplicativos e dispositivos uma interação mais concisa e integrada, ao mesmo tempo em que se beneficia de uma infraestrutura altamente personalizável e eficiente.

CAPÍTULO 1

Arquitetura

Este documento descreve a arquitetura atual que guia a implementação da *dojot*, detalhando os componentes que compõem a solução, assim como as suas funcionalidades e como cada um deles contribui para a plataforma como um todo.

Aqui é feita uma breve explicação dos componentes, sendo esta descrição em alto nível e sem o objetivo de explicar os detalhes de implemenação de cada um deles. Para isso, procure a documentação própria do componente.

Uma visão geral de toda a arquitetura é mostrada na figura acima e nas seções a seguir são fornecidos mais detalhes sobre cada componente.

Índice

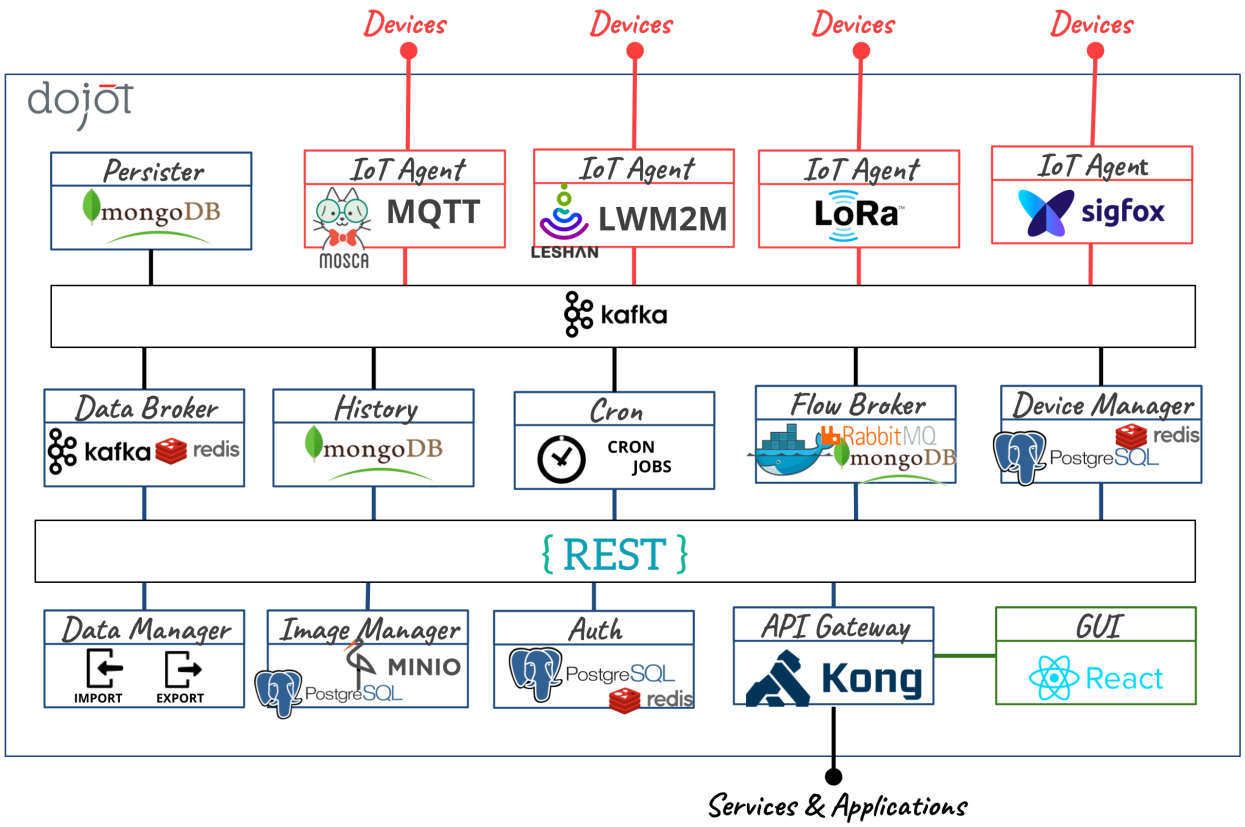


Fig. 1.1: A arquitetura de microsserviço da plataforma dojot.

1.1 Componentes

A *dojot* foi projetada para tornar possível uma prototipagem rápida, fornecendo uma plataforma fácil de usar, escalável e robusta. Sua arquitetura interna faz uso de muitos componentes conhecidos de código aberto e outros projetados e implementados pela equipe dojot.

Usando a dojot: um usuário configura dispositivos de IoT por meio da GUI ou diretamente usando as APIs REST fornecidas pelo API Gateway. Os fluxos de processamento de dados também podem ser configurados - essas entidades podem executar uma variedade de ações, como gerar notificações quando um atributo de dispositivo específico atingir um determinado limite ou salvar todos os dados gerados por um dispositivo em um banco de dados externo. À medida que os dispositivos começam a enviar suas leituras para dojot, um usuário pode:

- receba essas leituras em tempo real pelos canais socket.io;
- consolidar todos os dados em dispositivos virtuais;
- reunir todos os dados do banco de dados histórico e assim por diante.

Esses recursos podem ser usados por meio de APIs REST - esses são os blocos de construção básicos que qualquer aplicativo baseado em dojot deve usar. A GUI dojot fornece uma maneira fácil de executar operações de gerenciamento para todas as entidades relacionadas à plataforma (usuários, dispositivos, modelos e fluxos) e também pode ser usada para verificar se tudo está funcionando bem.

O contexto do usuário é isolado e não há compartilhamento de dados, as credenciais de acesso são validadas pelo serviço de autorização para cada operação (solicitação da API). Portanto, um usuário pertencente a um contexto específico (tenant) não pode acessar nenhum dado (incluindo dispositivos, modelos, fluxos ou quaisquer outros dados relacionados a esses recursos) de outros

Depois que os dispositivos são configurados, o IoT Agent é capaz de mapear os dados recebidos dos dispositivos, encapsulados no MQTT, por exemplo, e enviá-los ao intermediário de mensagens para distribuição interna. Dessa forma, os dados chegam ao serviço de persistência, por exemplo, para que possam persistir os dados em um banco de dados.

Para maiores informações sobre o que acontece na dojot, você pode conferir os *repositórios GitHub do projeto <https://github.com/dojot>*. Lá você encontrará todos os componentes utilizados pela plataforma.

Cada um dos componentes que compõem a arquitetura é brevemente descrito nas sessões subsequentes.

1.1.1 Kafka + DataBroker

O Apache Kafka é uma plataforma distribuída de mensageria que pode ser utilizada por aplicações que precisam transmitir dados ou consumir/produzir canais de dados. Em contraste com o Orion, um intermediador de contexto (context broker) utilizado na versão inicial da plataforma, o Kafka parece mais apropriado para assumir os requisitos arquiteturais da *dojot* (segregação de responsabilidades, simplicidade e assim por diante).

No Kafka, utiliza-se uma estrutura de tópicos especializada para garantir isolamento de dados de usuários e aplicações, viabilizando uma arquitetura multi-inquilino (*multi-tenant*).

O serviço DataBroker utiliza um banco de dados na memória para obter eficiência. Ele adiciona contexto ao Apache Kafka, possibilitando que serviços internos ou externos possam consumir dados em tempo-real com base no contexto. O DataBroker também é um serviço distribuído para evitar que seja um ponto único de falha ou mesmo um gargalo para a arquitetura.

1.1.2 DeviceManager

O DeviceManager é uma entidade central responsável por manter as estruturas de dados de dispositivos e modelos (templates). Também é responsável por publicar quaisquer atualizações para todos os componentes interessados através do Kafka.

O serviço não mantém estados e tem seus dados persistidos em banco de dados, onde suporta isolamento de dados por usuários e aplicações, viabilizando uma arquitetura de middleware com multi-tenancy.

1.1.3 Agente IoT

Um agente IoT é um serviço de adaptação entre dispositivos físicos e componentes principais da *dojot*. Pode ser entendido como um *driver de dispositivo* para um conjunto de dispositivos. A plataforma *dojot* pode ter vários agentes IoT, cada um deles especializado em um protocolo específico, como, por exemplo, MQTT/JSON, CoAP/LWM2M, Lora/ATC, Sigfox/WDN and HTTP/JSON.

O agente IoT também é responsável por garantir que a sua comunicação com dispositivos seja feita por meio de canais seguros.

1.1.4 Serviço de Autorização de Usuários

Serviço que implementa o gerenciamento de perfil de usuários e controle de acesso. Basicamente qualquer chamada de aplicação através do API Gateway é validada por este serviço.

Para ser capaz de atender a um grande volume de chamadas de autorização, faz uso de cache, não mantém estados e pode ser escalado horizontalmente. Seus dados são mantidos em banco de dados clusterizável.

1.1.5 Flowbroker (Flow builder)

Esse serviço provê mecanismos para construir fluxos de processamento de dados para execução de um conjunto de ações. Os fluxos podem ser estendidos usando um bloco de processamento externo (que pode ser incluído utilizando APIs REST).

1.1.6 Data Manager

Este serviço gerencia a configuração de dados do dojot, possibilitando importar e exportar configurações.

1.1.7 Cron

Cron é um microsserviço de um dojot que permite agendar eventos (ou requisições) a serem emitidos (feitas) para outros microsserviços dentro da plataforma dojot.

1.1.8 Persister/History

O componente Persister funciona como um condutor de dados e eventos que devemser persistidos em um banco de dados. Os dados são convertidos em uma estrutura de armazenamento que é enviada para o banco de dados correspondente.

Para armazenamento interno, utiliza-se uma base de dados não-relacional MongoDB que pode ser configurada em modo Sharded Cluster dependendo do caso de uso.

The persisted data can be queried through a Rest API provided by the History microservice.

1.1.9 Kong API Gateway

O Kong API Gateway é utilizado como um ponto de fronteira entre as aplicações e serviços externos e os serviços internos do dojot. Isso resulta em inúmeras vantagens como, por exemplo, ponto único de acesso e facilidade na aplicação de regras sobre as chamadas de APIs como limitação de tráfego e controle de acesso.

1.1.10 GUI

A Interface Gráfica de Usuário (GUI) na *dojot* é uma aplicação WEB que provê interfaces responsivas para gerenciamento da plataforma, incluindo funcionalidades como:

- **Gerenciamento de perfil de usuários**: permite definir perfis e quais APIs podem ou não ser acessadas pelo respectivo perfil.
- **Gerenciamento de usuários**: permite operações de criação, visualização, edição e remoção.
- **Gerenciamento de modelos de dispositivos**: operações de criação, visualização, edição e remoção.
- **Gerenciamento de dispositivos**: operações de criação, visualização (dispositivo e dados em tempo real), edição e remoção.
- **Gerenciamento de fluxos de processamento**: permite operações de criação, visualização, edição e remoção de fluxos de processamento de dados.
- **Notificações**: visualiza as notificações do sistema (em tempo real e histórico unificados)

1.1.11 Image manager

Este componente é responsável pelo armazenamento e recuperação de imagens de firmware de dispositivos.

1.2 Infraestrutura

Alguns outros componentes são utilizados na dojot, são eles:

- postgres: esse banco de dados é utilizado para persistir informações de vários componenentes, como do gerenciador de dispositivos.
- redis: é um banco de dados em memória usado como cache em vários componentes, como o serviço de osquestração, gerenciador de subscrição, agentes IoT e outros. É bem leve e fácil de usar.
- rabbitMQ: intermediador de mensagens utilizado no serviço de orquestração para implementar fluxos de ações relacionados que podem ser aplicados a mensagens recebidas dos componentes.
- Banco de dados mongo: solução de banco de dados amplamente utilizada que é fácil de usar e não adiciona esforço de acesso considerável (nos locais onde foi empregado na dojot).
- zookeeper: mantém sob controle serviços replicados em cluster.

1.3 Comunicação

Todos os componentes se comunicam de duas maneiras:

- Por meio de requisições HTTP: se um componente necessita recuperar dados de outro, como um agente IoT que precisa a lista de dispositivos configurados do gerenciador de dispositivos, ele pode enviar uma requisição HTTP para o componente apropriado.

- Por meio de mensagens Kafka: se um componente precisa enviar novas informações sobre um recurso controlado por ele (como novos dispositivos criados no gerenciador de dispositivos), o componente pode publicar esses dados através do Kafka. Utilizando esse mecanismo, qualquer outro componente que esteja interessado em tal informação precisa apenas ouvir um tópico específico para recebê-la. Note que este mecanismo não faz quaisquer associações difíceis entre componentes. Por exemplo, o gerenciador de dispositivos não sabe quais componentes precisam de suas informações e um agente IoT não necessita saber qual componente está enviando dados através de um tópico específico.

CAPÍTULO 2

Arquitetura do Agente IoT

Este documento descreve a arquitetura do agente IoT usada pelo dojot. Ele define um conjunto de recursos e opções básicas que devem ser seguidos para serem alinhados com a arquitetura dojot.

2.1 Quem deve ler isso

Desenvolvedores que desejam criar novos agentes de IoT para serem usados com dojot

2.2 Introdução

Usar dojot envolve lidar com as seguintes entidades:

- **dispositivos físicos**: dispositivos que enviam mensagens para agentes da IoT. Eles podem ter sensores e podem ser configuráveis, mas isso não é obrigatório. Além disso, eles devem ter algum tipo de conectividade com outros serviços, para que possam enviar suas leituras para esses serviços.
- **usuários**: quem envia solicitações ao dojot para gerenciar recursos, recuperar dados históricos do dispositivo, criar subscrição, gerenciar fluxos e assim por diante.
- **tenants**: separação lógica entre recursos que podem estar associados a vários usuários.
- **recursos**: elementos associados a uma entidade específica. Eles são:
  - *dispositivos*: representação de um elemento que possui atributos. Esse elemento pode ser um dispositivo físico ou virtual - um que não recebe atualizações de atributos diretamente por um dispositivo.
  - *modelos de dispositivo*: blueprints de dispositivos que contêm uma lista de atributos associados a essa classe de dispositivos. Todos os dispositivos são criados com base em um modelo, do qual herdarão atributos.
  - *tópicos*: Canais de comunicação Kafka usados para enviar e receber mensagens entre serviços dojot.
  - *flows*: Sequência de blocos de processamento criados por um usuário ou aplicativo e utilizados para analisar e pré-processar dados.

- **subjects**: grupo de tópicos que compartilham um fluxo de mensagens comum. Por exemplo, pode haver muitos tópicos usados para transmitir dados do dispositivo. Todos eles pertencem ao mesmo assunto "device-data".

Quando um novo agente de IoT é criado, todas essas entidades devem ser levadas em consideração de maneira coordenada. Este documento lista todos os requisitos básicos para um novo agente de IoT e eles são categorizados nos seguintes grupos:

1. **Segurança do dispositivo**: Os agentes de IoT devem poder verificar se uma conexão de dispositivo é válida ou não. Uma conexão de dispositivo válida é definida como uma originada por um dispositivo físico confiável (ou qualquer elemento representativo, como gateways) que tem permissão para se conectar ao agente de IoT. Um dispositivo é considerado confiável por: (1) criar um dispositivo associado a ele (que pode incluir informações de segurança, como chaves criptográficas) ou (2) indicar diretamente ao agente da IoT que um dispositivo ou um elemento representativo tem permissão para conectar-se a ele (para que os elementos que servem como conexões de relé possam ser usados de maneira adequada e segura).
2. **Separação de contexto de informações**: cada recurso (dispositivo, modelos, tópicos e fluxos) está associado a um *tenant* específico e as entidades que não pertencem a esse *tenant* não devem ter permissão para acessar seus recursos. Isso é válido em todo a dojot e não é uma exceção para agentes de IoT. Portanto, um agente de IoT deve tratar separadamente todos os dispositivos que pertencem a diferentes *tenants* - incluindo o fato de que ninguém em um *tenant* deve saber a existência de outros *tenants*. Por exemplo, um agente MQTT IoT não deve permitir que mensagens enviadas para seu broker de dispositivos associados ao *tenant* A sejam publicadas em dispositivos inscritos no mesmo tópico pertencente ao *tenant* B.
3. **Informações e gerenciamento de agentes de IoT**: qualquer agente de IoT deve publicar seus recursos e modelos de informação. Por exemplo, ele deve informar outros serviços sobre qual é o modelo de dispositivo aceito para receber e enviar mensagens corretamente para um dispositivo físico específico. Também deve oferecer uma interface de gerenciamento para que um usuário possa alterar e recuperar seu comportamento, como opções de log, estatísticas, cotas etc.
4. **Operação do agente IoT**:Os agentes de IoT devem poder receber e enviar mensagens (se permitido pelo protocolo) aos dispositivos e, portanto, enviar atualizações para outros serviços dojot com base nas mensagens recebidas do dispositivo. Todas as mensagens recebidas de um dispositivo específico e enviadas para outros serviços dojot devem ser enviadas na mesma ordem em que foram recebidas. Os agentes de IoT também devem poder ativar ou desativar o processamento de mensagens de um dispositivo específico e detectar o estado do dispositivo.

   Um recurso extra que um agente de IoT pode implementar são as atualizações de firmware. Dependendo do protocolo subjacente, pode ser possível fazer isso de maneira fácil, segura e confiável.

Cada um desses grupos será detalhado nas seções a seguir.

2.3 Segurança do dispositivo

Um gerenciamento IoT deve levar em consideração os seguintes aspectos da comunicação do dispositivo:

1. Identidade do dispositivo: deve aceitar apenas conexões de dispositivos físicos autorizados. A verificação se uma nova conexão foi originada por um dispositivo autorizado (que inclui verificar se um dispositivo específico está autorizado ou não) deve depender de chaves públicas e / ou certificados assinados.
2. Segurança do canal de comunicação: todas as mensagens trocadas com um dispositivo físico devem ser criptografadas usando padrões criptográficos conhecidos, como TLS. Quaisquer protocolos de segurança próprios devem ser evitados.
3. Revogação de certificado: o agente da IoT deve poder descartar qualquer mensagem do dispositivo autorizado anteriormente se seus dados de segurança tiverem sido comprometidos de alguma forma. Por exemplo, se a chave privada associada a um dispositivo específico vazar, todas as suas mensagens deverão ser ignoradas, pois não há garantia de que elas vieram desse dispositivo.

Cada um desses aspectos será detalhado nas seções a seguir.

2.3.1 Identidade do dispositivo

A verificação de identidade do dispositivo é o ponto de partida para lidar com a segurança da comunicação. Essa validação indicará ao agente IoT se o dispositivo que abriu a conexão é quem diz que é. Além disso, o agente IoT deve, uma vez que essa validação for bem-sucedida, verificar se este dispositivo pode se conectar a ele, verificando seu ID. Esta seção mostrará como fazer isso.

Para protocolos orientados a conexão, o agente IoT deve aceitar apenas conexões para dispositivos que possuem um certificado assinado por uma autoridade confiável pela dojot. Depois que esse certificado é válido, a identidade do dispositivo pode ser verificada de duas formas:

- ID do dispositivo codificado no certificado: embora seja um mecanismo menos confiável, ele permite maior flexibilidade usando muitos dispositivos em uma implantação controlada. Isso se baseia na configuração do nome comum (campo de certificado CN) como ID do dispositivo dojot. Portanto, o agente IoT deve verificar se este dispositivo existe ou não e permitir ou negar a conexão imediatamente, dependendo dessa verificação. Os pontos fracos desses mecanismos é que o certificado do dispositivo deve ser assinado pela CA interna do dojot (uma vez que existe um procedimento para assinar apenas um certificado por dispositivo) e, se esse certificado for válido, seu ID também deverá ser válido. Se qualquer outra autoridade de certificação for usada, esse mecanismo não terá uso válido.
- O agente IoT possui todos os certificados válidos: se um administrador deseja usar uma CA externa para assinar todos os certificados de dispositivo, não há controle real de qual ID de dispositivo foi usada para gerar um certificado específico. Portanto, o agente IoT deve ter todos os certificados válidos mapeados adequadamente em uma lista de dispositivos - isso garantirá que apenas um certificado seja permitido para um dispositivo específico e vice-versa.

Usando o primeiro mecanismo, o dispositivo (ou um operador que configura um dispositivo pela primeira vez) deve chamar a CA dojot para gerar um certificado assinado para si. Não há nenhuma ação adicional para o agente IoT usar CA da dojot.

O segundo mecanismo, no entanto, exige que um agente de IoT ofereça métodos para gerenciar certificados. O desenvolvedor deve levar em conta também que esse agente de IoT deve poder escalar - esses certificados devem estar acessíveis a todas as instâncias do agente de IoT, se permitido pela implantação.

2.3.2 Segurança da comunicação

Com um certificado válido, um dispositivo pode criar um canal de comunicação com dojot. Para canais orientados a conexão, esse certificado deve ser usado juntamente com chaves criptográficas para fornecer um canal criptografado. Para outros tipos de canal (como canais para troca de mensagens por meio de um gateway, como LoRa ou sigfox), basta garantir que a conexão entre o dojot e o servidor de back-end seja segura. A identidade do back-end deve ser declarada previamente. Uma vez que se sabe que é confiável, todas as suas mensagens podem ser processadas sem grandes preocupações.

2.3.3 Revogação de certificado

Um agente de IoT deve poder ser informado sobre certificados revogados. Ele deve expor uma API ou mensagens de configuração para permitir isso. Não deve permitir nenhuma comunicação com um dispositivo específico que usa um certificado revogado.

2.4 Separação de contexto de informações

Um *tenant* pode ser pensado simplesmente como um grupo de usuários que compartilham alguns recursos. Mas seu significado pode ir além disso - pode implicar que esses recursos não compartilhem nenhuma infraestrutura comum (considerando qualquer coisa que transmita, processe ou armazene dados) com recursos pertencentes a outros *tenants*. Pode-se ter instâncias de software separadas para processar dados de *tenants* diferentes, para que o processamento de dados de um *tenant* não afete o processamento de dados do outro, atingindo um nível mais alto de separação de contexto.

Embora isso seja desejável, alguns cenários de implantação podem forçar o uso de parte da mesma infraestrutura para diferentes *tenants* (por exemplo, quando a implantação tem um número reduzido de unidades de processamento ou conexões de rede). Portanto, para ter uma separação mínima de contexto entre os *tenants*, um agente de IoT deve usar tudo o que puder para separá-los, como diferentes threads, filas, soquetes etc., e não deve confiar apenas nos recursos dos cenários de implantação (como agentes IoT diferentes para diferentes *tenants*). Por exemplo, para protocolos baseados em tópicos, como o MQTT, pode-se forçar tópicos diferentes para diferentes inquilinos. Se um dispositivo publicar dados em um tópico específico de propriedade de outro *tenant*, essa mensagem será ignorada ou bloqueada (o envio de um erro de volta ao dispositivo pode ser um comportamento opcional). Portanto, nenhum dispositivo de um *tenant* pode enviar mensagens para qualquer dispositivo de outro *tenant*.

O mecanismo pelo qual a separação de contexto é implementada depende muito de qual protocolo é usado. Uma análise completa deve ser realizada para implementar adequadamente esse recurso.

2.5 Informações e gerenciamento de agentes de IoT

Um agente de IoT deve expor todas as informações necessárias para usá-lo corretamente. Deve expor:

- **Modelo de dispositivo**: um agente de IoT deve publicar qual é o modelo de dados aceito para um dispositivo válido. Isso deve ser feito publicando um novo modelo de dispositivo em outros serviços dojot. Deve haver um mecanismo para que instâncias diferentes do mesmo agente de IoT publiquem o mesmo modelo de dispositivo (incluindo quaisquer IDs de modelo). Se o modelo do dispositivo for atualizado em uma versão mais recente de um agente de IoT, o ID do modelo do dispositivo deverá mudar.
- **APIs de gerenciamento**: um agente de IoT deve ser gerenciável e deve expor suas APIs para isso. O conjunto mínimo de APIs de gerenciamento que um agente de IoT deve oferecer são:
  - *Log*: deve haver uma maneira de alterar o nível de log de um agente de IoT;
  - *Estatísticas*: um agente de IoT pode expor uma API para permitir que um usuário ou aplicativo recupere informações estatísticas sobre sua execução. Um administrador pode querer ativar ou desativar a geração de uma variável estatística específica, como o tempo de processamento.

Um agente de IoT também deve ser capaz de coletar informações estatísticas relacionadas à sua execução. Além disso, deve permitir que um administrador defina cotas para essas quantidades medidas. Essas quantidades podem incluir, mas não estão limitadas a:

- estatísticas de transmissão
  - number of received device messages from device (total, per device, per tenant)
  - número de mensagens recebidas do dispositivo (total, por dispositivo, por *tenant*)
  - número de mensagens enviadas aos dispositivos (total, por dispositivo, por *tenant*)
  - [opcional] tempo decorrido entre o recebimento de uma mensagem de um dispositivo físico e a publicação (total - média, por dispositivo - média, por *tenant* - média)
- Verificação de integridade do serviço do agente IoT - estatísticas do sistema (memória, disco etc.) usadas pelo serviço

Muitos outros valores podem ser reunidos. A lista acima é a lista mínima que um agente de IoT deve expor a outros serviços. Especialmente para verificação de integridade, há um documento detalhando como expô-lo.

2.6 Operação do agente IoT

O principal objetivo de um agente de IoT é publicar dados de um dispositivo específico em outros serviços dojot. Sua operação é dupla: receba e processe mensagens relacionadas ao gerenciamento de dispositivos de outros serviços, bem como receba mensagens dos próprios dispositivos (ou de seus elementos representativos) e publique esses dados em outros serviços.

As seções a seguir descrevem como um agente de IoT pode enviar e receber mensagens de/para outros serviços da dojot e quais são as considerações que ele deve levar em consideração ao receber mensagens de dispositivos físicos.

2.6.1 Mensagens

At start, all IoT agents (in fact, all services that need to receive or send messages related to devices) must know the list of configured tenants. This is the most basic piece of information that IoT agent needs to know in order to work properly. The request that should be sent to Auth service is this (all requests sent from dojot services to its own services should use the 'dojot-management' user):

Host: Auth	
Endpoint: /admin/tenants	Method: GET
Request	
Headers	Authorization: Bearer ${JWT}
Response	
Headers	Content-Type: application/json
Body format	`tenants => *tenant` `  tenant => string`

Um exemplo de resposta para esta requisição

```
{
  "tenants": [
    "admin",
    "users",
    "system"
  ]
}
```

Com esta lista, o agente IoT pode solicitar tópicos para receber eventos do ciclo de vida do dispositivo e do tenant e para publicar novos dados de atributo do dispositivo. Isso é feito enviando a seguinte solicitação ao DataBroker:

Host: DataBroker	
Endpoint: /topic/{subject}	Method: GET
Request	
Headers	Authorization: Bearer ${JWT}
Response	
Headers	Content-Type: application/json
Body format	`topic => string`

Um exemplo de resposta para esta requisição

```
{
  "topic": "c9b2c688-9e40-4032-877a-3d262acba9d0"
}
```

Alguns assuntos são "sensíveis ao tenant" (um tópico diferente será retornado para diferentes tenants) e outros não (o mesmo tópico será retornado independentemente do tenant). O DataBroker usará o tenant contido no token de autorização ao lidar com assuntos sensíveis ao tenant.

Os seguintes subjects devem ser usados pelo agente IoT

- dojot.tenancy
- dojot.device-manager.device-template
- dojot.device-manager.device
- device-data

Cada um desses aspectos será detalhado nas seções a seguir.

dojot.tenancy

O tópico relacionado a este assunto será usado para receber eventos do ciclo de vida do tenant. Sempre que um novo tenant for criado ou excluído, a seguinte mensagem será publicada:

Subject: dojot.tenancy	
Body format (JSON)	`type="CREATE"/"DELETE"` `tenant=>string`

Este assunto não é sensível ao tenant. Uma mensagem de exemplo recebida por este tópico é:

```
{
  "type": "CREATE",
  "tenant": "new_tenant"
}
```

dojot.device-manager.device-template

Subject: dojot.device-manager.device-template	

Body format (JSON)	event => "create"
	data => id label attrs
	  id => string
	  label => string
	  attrs => [*template_attrs]

dojot.device-manager.devices

o tópico relacionado a este assunto será usado para receber eventos do ciclo de vida do dispositivo para um tenant específico. Seu formato é:

Subject: dojot.device-manager.device	

Body format (JSON)	event => "create" / "update"
	meta => service
	  service => string
	data => id label templates attrs created
	  id => string
	  label => string
	  templates => *number
	  attrs => [*template_attrs]
	  created => iso_date

Body format (JSON)	event => "remove"
	meta => service
	  service => string
	data => id
	  id => string

Body format (JSON)	event => "actuate"
	meta => service
	  service => string
	data => id
	  id => string
	  attrs => *device_attrs

O template_attrs é um JSON de chave/valor simples com o ID do modelo como chave e a seguinte estrutura como valor:

```
{
  "template_id": "1",
  "created": "2018-01-05T15:41:54.840116+00:00",
  "label": "this-is-a-sample-attribute",
  "value_type": "float",
  "type": "dynamic",
  "id": 1
}
```

O atributo device_attrs é um JSON de chave/valor ainda mais simples, como:

```
{
  "temperature" : 10,
  "height" : 280
}
```

Este assunto é sensível ao tenant.

Uma mensagem de exemplo recebida por este tópico é:

```
{
  "event": "create",
  "meta": {
    "service": "admin"
  },
  "data": {
    "id": "efac",
    "label": "Device 1",
    "templates": [1, 2, 3],
    "attrs": {
      "1": [
        {
          "template_id": "1",
          "created": "2018-01-05T15:41:54.840116+00:00",
          "label": "this-is-a-sample-attribute",
          "value_type": "float",
          "type": "dynamic",
          "id": 1
        }
      ]
    },
    "created": "2018-02-06T10:43:40.890330+00:00"
  }
}
```

device-data

O tópico relacionado a este assunto será usado para publicar dados recuperados de um dispositivo físico em outros serviços dojot. Seu formato é:

Subject: device-data	

Body format (JSON)	`metadata => deviceid tenant timestamp recv_time`
	`  deviceid => string`
	`  tenant => string`
	`  timestamp => unix_timestamp`
	`  recv_time => unix_timestamp`
	`attrs => *device_attrs`

Esse assunto é sensível ao tenant. O registro de data e hora está associado ao momento em que os valores do atributo foram coletados pelo dispositivo (isso pode ser feito pelo próprio dispositivo ou pelo agente IoT, se nenhum registro de data e hora foi definido pelo dispositivo). O atributo recv_time indica quando a mensagem foi recebida.

Uma mensagem de exemplo recebida por este tópico é:

```
{
  "metadata": {
    "deviceid": "c6ea4b",
    "tenant": "admin",
    "timestamp": 1528226137452,
    "recv_time": 1528226137462
  },
  "attrs": {
    "humidity": 60
  }
}
```

2.6.2 Atualização de Firmware

Um agente de IoT pode implementar mecanismos para atualizar o firmware nos dispositivos.

2.7 Comportamento

A ordem na qual um dispositivo físico envia seus atributos não deve ser alterada quando o agente da IoT publica esses dados em outros serviços dojot.

Se o protocolo impuser qualquer ID único externo para cada dispositivo, o agente de IoT deverá criar uma tabela de correlação para converter adequadamente esse ID único externo em ID de dispositivo dojot e vice-versa.

2.8 Bibliotecas para auxiliar no desenvolvimento de novos IotAgents

Temos bibliotecas em node.js **recomendado** (https://github.com/dojot/iotagent-nodejs) e java (https://github.com/dojot/iotagent-java) parafacilitar o desenvolvimento de um IotAgent.

CAPÍTULO 3

Conceitos

Este documento fornece informações sobre os conceitos e abstrações da dojot.

Índice

Nota:

- **Público**
  - Usuários que desejam dar uma olhada em como a dojot funciona;
  - Desenvolvedores de aplicativos.
- Nível: básico

3.1 noções básicas da dojot

Antes de usar a dojot, você deve estar familiarizado com algumas operações e conceitos básicos. Eles são muito simples de entender e usar, mas sem eles, todas as operações podem se tornar obscuras e sem sentido.

Na próxima seção, há uma explicação de algumas entidades básicas no dojot: dispositivos, modelos e fluxos. Com esses conceitos em mente, apresentamos um pequeno tutorial sobre como usá-los na dojot - ele cobre apenas o acesso à API. Existe um tutorial orientando como utilizar a interface WEB (GUI) em *Usando a interface WEB*.

Se você quiser obter mais informações sobre como o dojot funciona internamente, consulte a *Arquitetura* para se familiarizar com todos os componentes internos.

3.1.1 Autenticação de usuário

Todas as solicitações HTTP suportadas pela dojot são enviadas para o gateway da API. Para controlar qual usuário deve acessar quais terminais e recursos, a dojot utiliza o JSON Web Token (uma ferramenta útil é o jwt.io) que codifica coisas como (não se limitando a eles):

- Identidade do usuário
- Dados de validação
- Data de validade do token

O componente responsável pela autenticação do usuário é auth. Você pode encontrar um tutorial de como autenticar um usuário e obter um token de acesso na auth documentation.

3.1.2 Dispositivos e modelos

Na dojot, um dispositivo é uma representação digital de um dispositivo ou gateway real com um ou mais sensores ou de um virtual com sensores/atributos inferidos de outros dispositivos. Em toda a documentação, esse tipo de dispositivo será chamado simplesmente de "dispositivo". Se o dispositivo real precisar ser referenciado, nós o chamaremos de "dispositivo físico".

Considere, por exemplo, um dispositivo físico com sensores de temperatura e umidade; ele pode ser representado na dojot como um dispositivo com dois atributos (um para cada sensor). Chamamos esse tipo de dispositivo como dispositivo normal ou por seu protocolo de comunicação, por exemplo, dispositivo MQTT ou dispositivo CoAP.

Também podemos criar dispositivos que não correspondem diretamente aos seus homólogos físicos, por exemplo, podemos criar um com maior nível de informação de temperatura (está ficando mais quente ou mais frio) cujos valores são inferidos a partir de sensores de temperatura de outros dispositivos. Esse tipo de dispositivo é chamado de dispositivo virtual.

Todos os dispositivos são criados com base em um modelo, que pode ser pensado como um modelo de dispositivo. Como "modelo", poderíamos pensar em números de peça ou modelos de produtos - um protótipo a partir do qual os dispositivos são criados. Os modelos na dojot têm um rótulo (qualquer sequência alfanumérica), uma lista de atributos que conterão todas as informações emitidas pelo dispositivo e, opcionalmente, alguns atributos especiais que indicarão como o dispositivo se comunica, incluindo métodos de transmissão (protocolo, portas, etc.) e formatos de mensagem.

De fato, os modelos podem representar não apenas "modelos de dispositivos", mas também podem abstrair uma "classe de dispositivos". Por exemplo, poderíamos ter um modelo para representar todos os termômetros que serão usados na dojot. Este modelo teria apenas um atributo chamado, digamos, "temperatura". Ao criar o dispositivo, o usuário selecionaria seu "modelo físico", digamos TexasInstr882 e o modelo "termômetro". O usuário também teria que adicionar instruções de tradução (implementadas em termos de fluxos de dados, construídas no construtor de fluxo) para mapear a leitura de temperatura que será enviada do dispositivo para um atributo de "temperatura".

Para criar um dispositivo, um usuário seleciona quais modelos irão compor esse novo dispositivo. Todos os seus atributos são mesclados e associados a ele - eles estão fortemente vinculados ao modelo original para que qualquer atualização de modelo reflita todos os dispositivos associados.

O componente responsável pelo gerenciamento de dispositivos (reais e virtuais) e modelos é o DeviceManager.

A DeviceManager documentação explica mais detalhadamente todas as operações disponíveis.

3.1.3 Flows

Um fluxo é uma sequência de blocos que processa um evento ou mensagem de dispositivo específica. Contém:

- ponto de entrada: um bloco representando qual é o gatilho para iniciar um fluxo específico;
- blocos de processamento: um conjunto de blocos que executam operações usando o evento. Esses blocos podem ou não usar o conteúdo desse evento para processá-lo ainda mais. As operações podem ser: testar conteúdo para valores ou intervalos específicos, análise de posicionamento geográfico, alterar atributos de mensagens, executar operações em elementos externos e assim por diante.
- ponto de saída: um bloco que representa para onde os dados resultantes devem ser encaminhados. Esse bloco pode ser um banco de dados, um dispositivo virtual, um elemento externo e assim por diante.

O componente responsável por lidar com esses fluxos é flowbroker

CAPÍTULO 4

Componentes e APIs

4.1 Components

Tabela 4.1: Components

Componentes	Repositório / Site principal	Documentação para Componentes	API de componentes Documentação
MongoDB	MongoDB site	MongoDB doc.	
Postgres	PostgreSQL site	PostgreSQL doc.	
Kong API gateway (Community Edition)	Kong site	Kong doc.	
Redis	Redis site	Redis doc.	
Zookeeper	Zookeeper site	Zookeeper doc.	
Kafka	Kafka site	Kafka doc.	
Auth	GitHub - auth	Auth doc.	API - auth
History	GitHub - history		API - history
Device Manager	GitHub - DeviceManager	DeviceManager doc.	API - DeviceManager
Image Manager	GitHub - image-manager		API - image-manager
GUI	GitHub - GUI		
Flowbroker	GitHub - flowbroker		API - flowbroker
Databroker	GitHub - data-broker		API - data-broker
Iotagent Mosca (MQTT)	GitHub - iotagent-mosca		
EJBCA-REST	GitHub - EJBCA-REST		API - EJBCA-REST
Data Manager	GitHub - Data Manager		API - Data Manager
Cron	GitHub - Cron		API - Cron

4.2 APIS expostas (API Gateway)

O API gateway usado na dojot reencaminha alguns dos *endpoints* dos componentes. A tabela a seguir mostra quais **endpoint exposto pelo API gateway** são mapeado para quais **endpoint dos componentes**, e suas **endpoint componentes documentações** usado via API Gateway. Veja mais como usar as APIS em Using API interface.

Tabela 4.2: *Endpoint* esposto

Componentes	*Endpoint* esposto pelo API *gateway*	Componentes *Endpoint*	Componentes *Endpoint* Documentação	Precisa de Autenticação
GUI	/	/		Não
Device Manager	/device	/device	API - DeviceManager	Sim
Device Manager	/template	/template	API - DeviceManager	Sim
Flowbroker	**/flows**	/	API - flowbroker	Sim
Auth	**/auth**	/	API - auth	Não
Auth	**/auth**/revoke	/revoke		Não
Auth	**/auth**/user	/user	API - auth	Sim
Auth	**/auth**/pap	/pap	API - auth	Sim
History	**/history**	/	API - history	Sim
EJBCA REST	/sign	/sign	API - EJBCA-REST	Sim
EJBCA REST	/ca	/ca	API - EJBCA-REST	Sim
EJBCA REST	/user	/user	API - EJBCA-REST	Sim
Data Manager	/import	/import	API - Data Manager	Sim
Data Manager	/export	/export	API - Data Manager	Sim
Cron	/cron	/cron	API - Cron	Sim
Image Manager	**/fw-image**	/	API - image-manager	Sim
Data Broker	/device/ {deviceID} /latest	/device/ {deviceID} /latest		Sim
Data Broker	/subscription	/subscription		Sim
Data Broker	/stream	/stream		Sim
Data Broker	/socket.io	/socket.io	API - data-broker	Não

NOTA: Alguns endpoints não são expostos, mas são utilizados internamente.

Além disso, o gateway da API redireciona os *endpoints* com suas repectivas portas do componente, para que se tornem uniformes: todos eles são acessíveis pela mesma porta (o padrão é a porta TCP 8000), consulte a tabela a seguir.

Tabela 4.3: *Endpoints* originais para o API gateway

Componentes	Endpoints Originais	Gateway Endpoint
GUI	host:80/	host:8000/
Device Manager	host:5000/device	host:8000/device
Device Manager	host:5000/template	host:8000/template
Flowbroker	host:80/	host:8000/flows
Auth	host:5000/	host:8000/auth
Auth	host:5000/auth/revoke	host:8000/auth/revoke
Auth	host:5000/user	host:8000/auth/user
Auth	host:5000/pap	host:8000/auth/pap
History	host:8000/	host:8000/history
EJBCA REST	host:5583/sign	host:8000/sign
EJBCA REST	host:5583/ca	host:8000/ca
Data Manager	host:3000/import	host:8000/import
Data Manager	host:3000/export	host:8000/export
Cron	host:5000/cron	host:8000/cron
Image Manager	host:5000/	host:8000/fw-image
Data Broker	host:80/device/{{deviceID}}/latest	host:8000/device/{deviceID}/latest
Data Broker	host:80/subscription	host:8000/subscription
Data Broker	host:80/stream	host:8000/stream
Data Broker	host:80/socket.io	host:8000/socket.io

4.3 Mensagens do Kafka

Essas são as mensagens enviadas pelos componentes e seus assuntos. Se você estiver desenvolvendo um novo componente interno (como um novo agente de IoT),consulte API - data-broker para verificar como receber mensagens enviadas por outros componentes na dojot.

Tabela 4.4: Endpoints Originais

Componentes	Mensagem	Subject
DeviceManager	Dispositivo CRUD (Criar, Ler, Atualizar e Excluir) (Mensagens - DeviceManager)	`dojot.device-manager.device`
iotagent-mosca	Atualização de dados do dispositivo (Mensagens - iotagent-mosca)	`device-data`
auth	Criação/remoção de *Tenants* (Mensagens - auth)	`dojot.tenancy`

CAPÍTULO 5

Comunicação interna

Esta página descreve como cada serviço na dojot se comunica.

5.1 Componentes

Os principais componentes atuais no dojot são mostrados em Fig. 5.1.

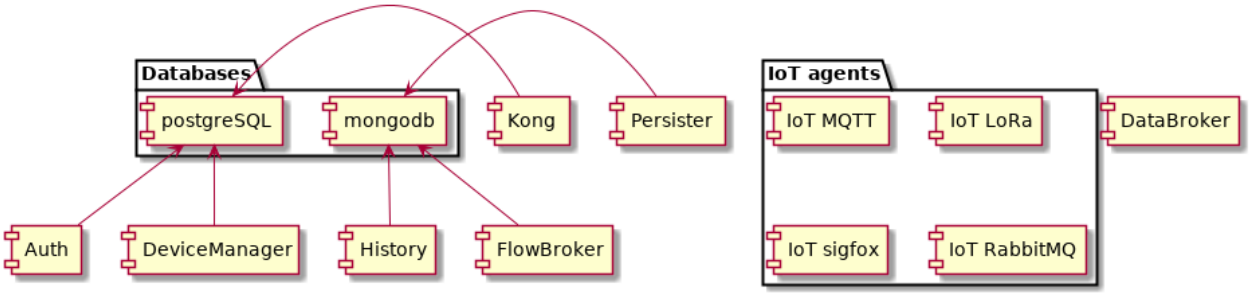


Fig. 5.1: Componentes da Dojot

Eles são:

- Auth: mecanismo de autenticação
- DeviceManager: armazenamento de dispositivo e modelo.
- Persister: componente que armazena todos os dados gerados pelo dispositivo.
- History: componente que expõe todos os dados gerados pelo dispositivo.
- DataBroker: lida com assuntos e tópicos do Kafka, além de conexões socket.io.
- Flowbroker: lida com fluxos (CRUD e execução de fluxo)
- IoT agents: agentes para diferentes protocolos.

Cada serviço será descrito brevemente nesta página. Mais informações podem ser encontradas na documentação de cada componente.

5.2 Mensagens e autenticação

Existem dois meios pelos quais os componentes dojot podem se comunicar: via solicitações HTTP REST e via Kafka. Eles são destinados a diferentes propósitos.

As solicitações HTTP podem ser enviadas no momento da inicialização quando um componente deseja, por exemplo, informações sobre recursos específicos, como lista de dispositivos ou *tenants*. Para isso, eles devem saber qual componente possui qual recurso para recuperá-los corretamente. Isso significa - e isso é muito importante porque conduz as escolhas arquiteturais na dojot - que apenas um único serviço é responsável por recuperar modelos de dados para um recurso específico (observe que um serviço pode ter várias instâncias). Por exemplo, o DeviceManager é responsável por armazenar e recuperar o modelo de informações para dispositivos e modelos, FlowBroker para descrições de fluxo, Histórico para dados históricos e assim por diante.

Kafka, por outro lado, permite uma comunicação pouco acoplada entre instâncias de serviços. Isso significa que um produtor (quem envia uma mensagem) não sabe quais componentes receberão sua mensagem. Além disso, qualquer consumidor não sabe quem gerou a mensagem de que está sendo ingerido. Isso permite que os dados sejam transmitidos com base em "interesses": um consumidor está interessado em receber mensagens com um determinado assunto (mais sobre isso mais tarde) e os produtores enviarão mensagens para todos os componentes que estiverem interessados nele. Observe que esse mecanismo permite que vários serviços emitam mensagens com o mesmo "assunto", bem como vários serviços que ingerem mensagens com o mesmo "assunto", sem soluções alternativas complicadas.

5.2.1 Enviando solicitações HTTP

Para enviar solicitações via HTTP, um serviço deve criar um token de acesso, descrito aqui. Não há outras considerações além de seguir a descrição da API associada a cada serviço. Isso pode ser visto na figura Fig. 5.2. Observe que todas as interações descritas aqui são abstrações das reais. Além disso, deve-se notar que essas interações são válidas apenas para componentes internos. Qualquer serviço externo deve usar o Kong como ponto de entrada.

Nesta figura, um cliente recupera um token de acesso para o administrador do usuário cuja senha é p4ssw0rd. Depois disso, um usuário pode enviar uma solicitação para as APIs HTTP usando-o. Isso é mostrado na Fig. Fig. 5.3. Nota: o mecanismo de autorização real é detalhado *Auth + API gateway (Kong)*.

Nesta figura, um cliente cria um novo dispositivo usando o token recuperado em Fig. 5.2. Essa solicitação é analisada por Kong, que chamará Auth para verificar se o usuário definido no token tem permissão para `POST` para o terminal `/device`. Somente após a aprovação dessa solicitação, Kong a encaminhará para o DeviceManager.

5.2.2 Enviando mensagens via Kafka

Kafka usa uma abordagem bem diferente. Cada mensagem deve ser associada a um assunto e um inquilino. Isso é mostrado na Fig. 5.4;

Neste exemplo, o DeviceManager precisa publicar uma mensagem sobre um novo dispositivo. Para isso, ele envia uma solicitação ao DataBroker, indicando qual *tenant* (dentro do token JWT) e qual assunto (`dojot.device-manager.devices`) deseja usar para enviar a mensagem. O DataBroker chamará o Redis para verificar se este tópico já foi criado e se o administrador dojot criou um perfil para essa tupla específica `{tenant, subject}`.

Os dois esquemas de perfis disponíveis são mostrados na Fig. 5.5 e na Fig. 5.6.

O esquema automático define o número de partições Kafka a serem usadas para o tópico que está sendo criado, bem como o fator de replicação (quantas réplicas estarão disponíveis para cada partição de tópico). Cabe a Kafka decidir

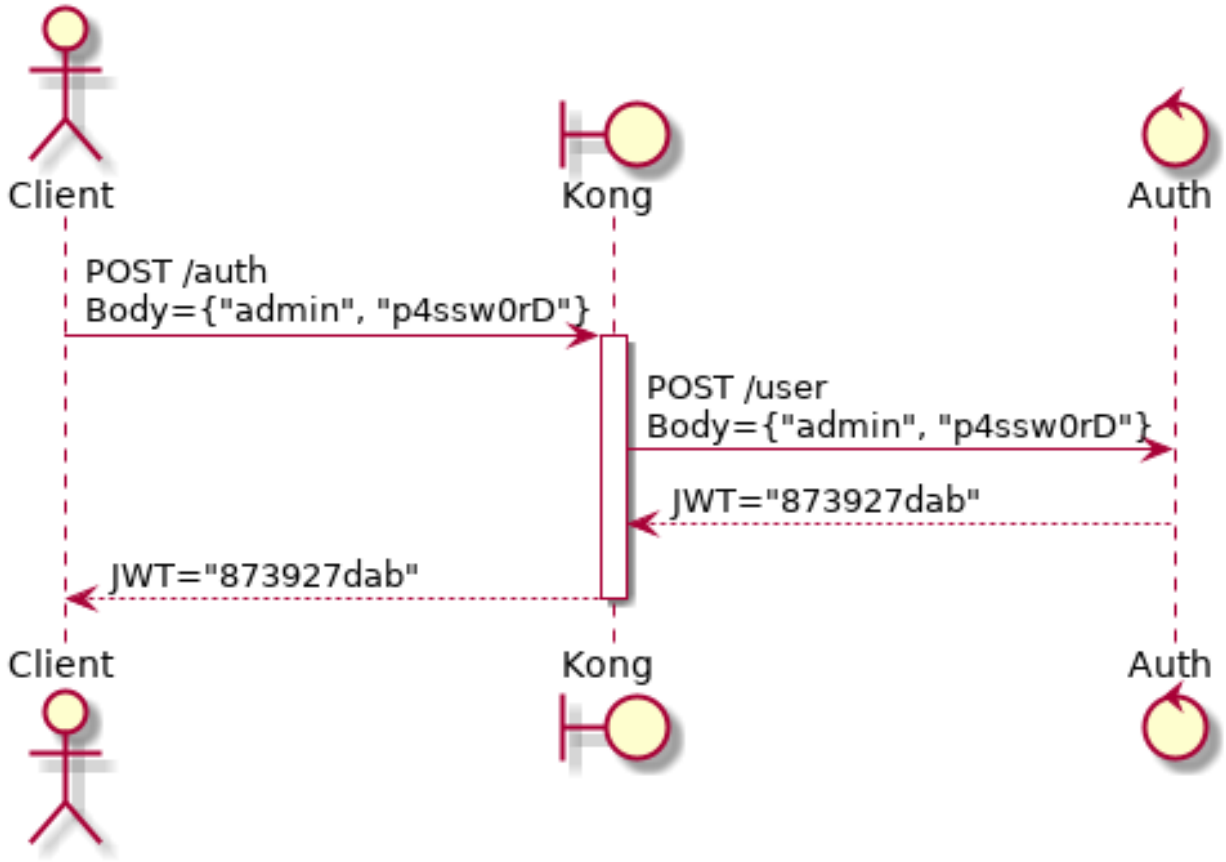


Fig. 5.2: Autenticação inicial

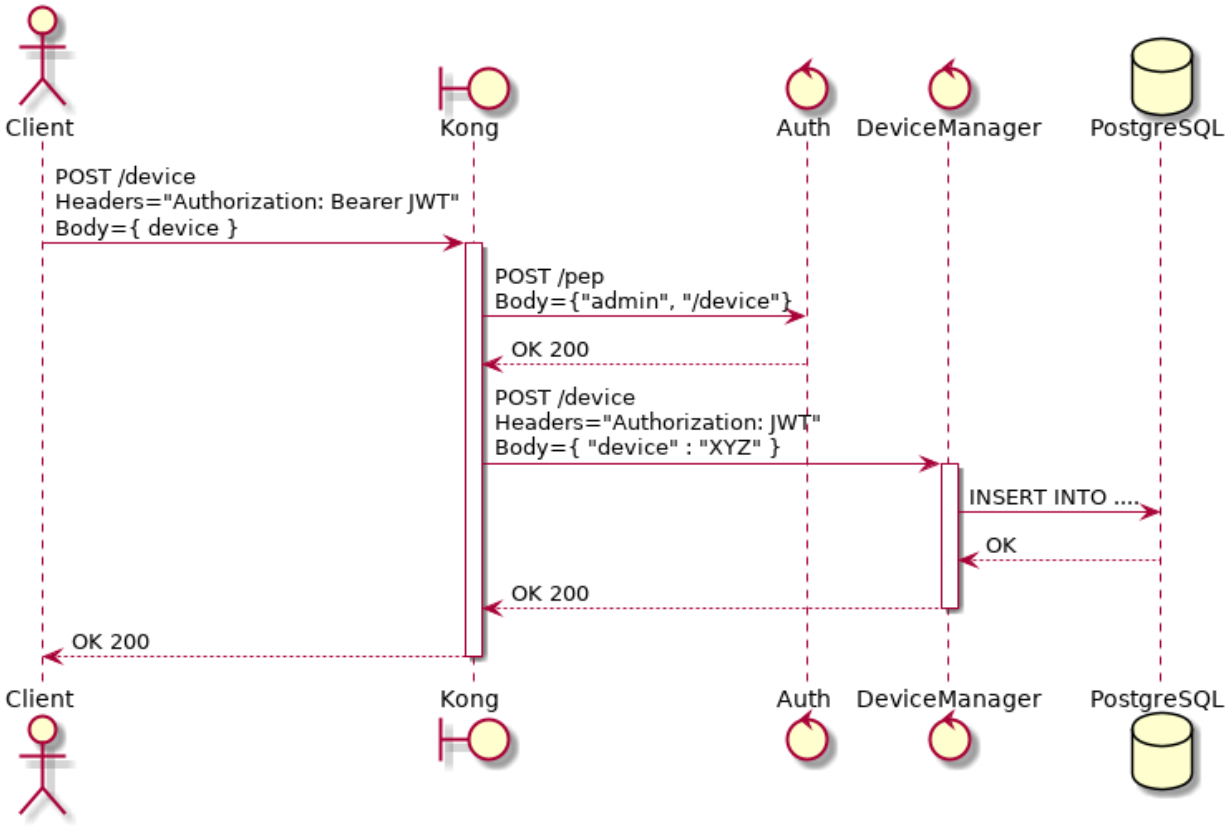


Fig. 5.3: Enviando mensagens para a API HTTP

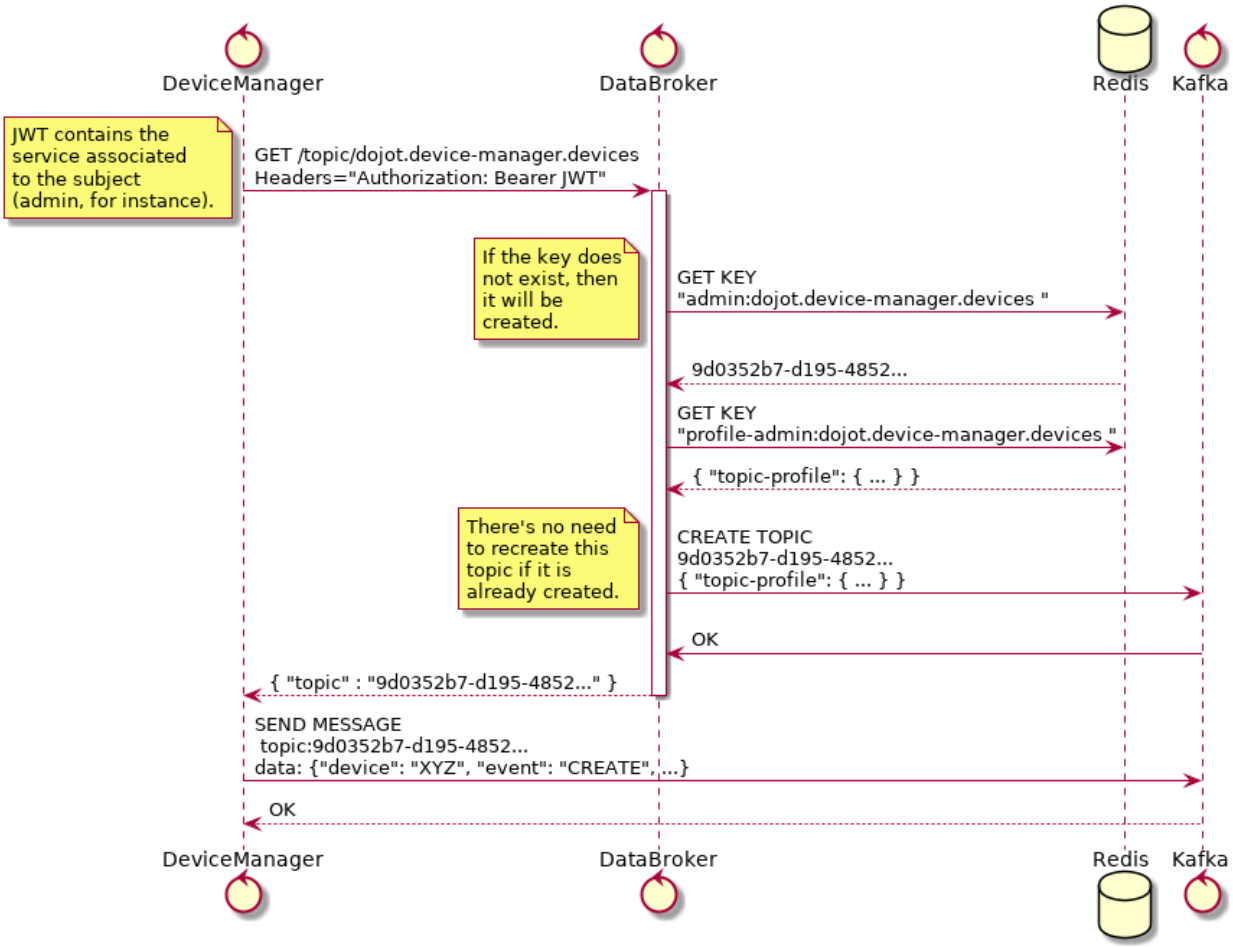


Fig. 5.4: Recuperando tópicos do Kafka

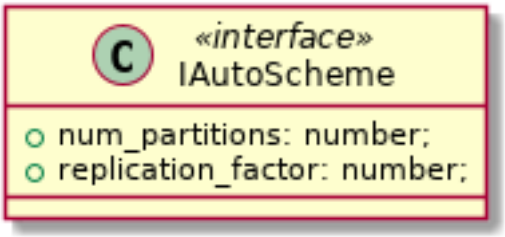


Fig. 5.5: Perfil de esquema automático

qual partição e réplica será atribuída a qual instância do broker. Você pode verificar Kafka partitions and replicas para conhecer um pouco mais sobre partição e réplicas. Claro que você pode conferir a Kafka's official documentation.

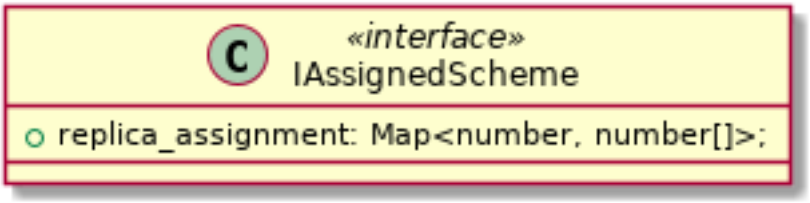


Fig. 5.6: Perfil de esquema atribuído

O esquema atribuído indica qual partição será alocada para qual instância Kafka. Isso inclui também réplicas (partições com mais de uma instância Kafka associada).

5.2.3 Inicialização dos *tenants*

Todos os componentes estão interessados em um conjunto de assuntos, que serão usados para enviar ou receber mensagens de Kafka. Como a dojot agrupa tópicos do Kafka e tenants em assuntos (um assunto será composto por um ou mais tópicos Kafka, cada um transmitindo mensagens para um tenant específico), o componente deve iniciar cada tenant antes de enviar ou receber mensagens. Isso é feito em duas fases: tempo de inicialização do componente e tempo de execução do componente.

Na primeira fase, um componente solicita ao Auth para recuperar todos os *tenants* configurados no momento. Está interessado, digamos, em consumir mensagens dos assuntos device-data e dojot.device-manager.devices. Portanto, ele solicitará ao DataBroker um tópico para cada *tenant* para cada assunto. Com essa lista de tópicos, ele pode criar Produtores e Consumidores para enviar e receber mensagens através desses tópicos. Isso é mostrado na Fig. 5.7.

A segunda fase inicia após a inicialização e seu objetivo é processar todas as mensagens recebidas pelo Kafka. Isso incluirá qualquer *tenant* criado após todos os serviços estarem em funcionamento. Fig. 5.8 mostra como lidar com essas mensagens.

Todos os serviços que estão de alguma forma interessados em usar assuntos devem executar este procedimento para receber corretamente todas as mensagens.

5.3 Auth + API gateway (Kong)

Auth é um serviço profundamente conectado ao Kong. É responsável pelo gerenciamento, autenticação e autorização do usuário. Como tal, é invocado por Kong sempre que uma solicitação é recebida por um de seus pontos de um dos endpoints. Esta seção detalha como isso é realizado e como eles funcionam juntos.

5.3.1 Configuração do Kong

Existem dois procedimentos de configuração ao iniciar o Kong na dojot:

1. Migrando Dados Existentes
2. Registrando endpoints e plug-ins de API.

A primeira tarefa é realizada simplesmente invocando Kong com uma bandeira especial.

O segundo é executado executando um script de configuração kong.config.sh. Seu único objetivo é registrar endpoints no Kong, como:

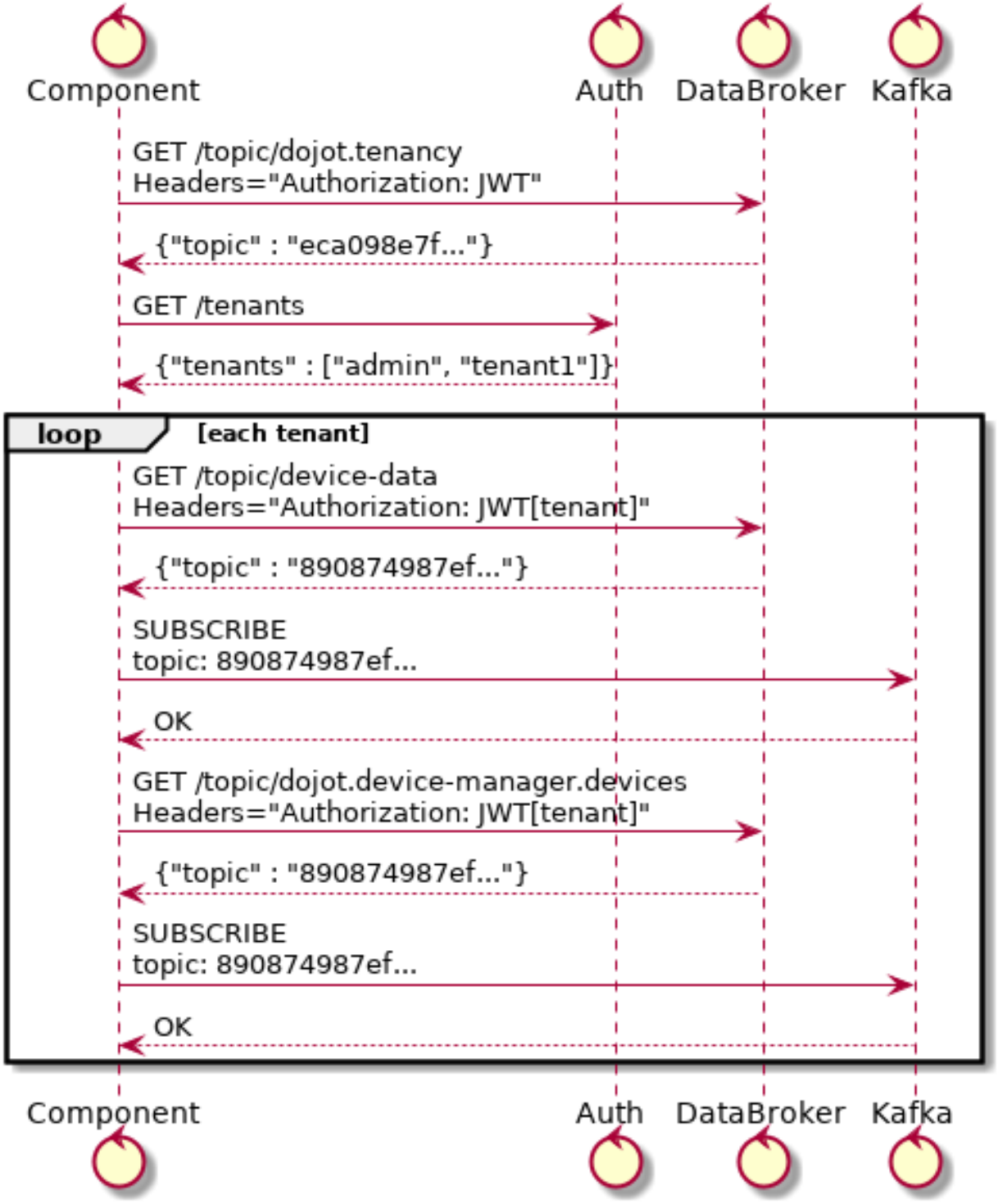


Fig. 5.7: Inicialização do *tenants* no início

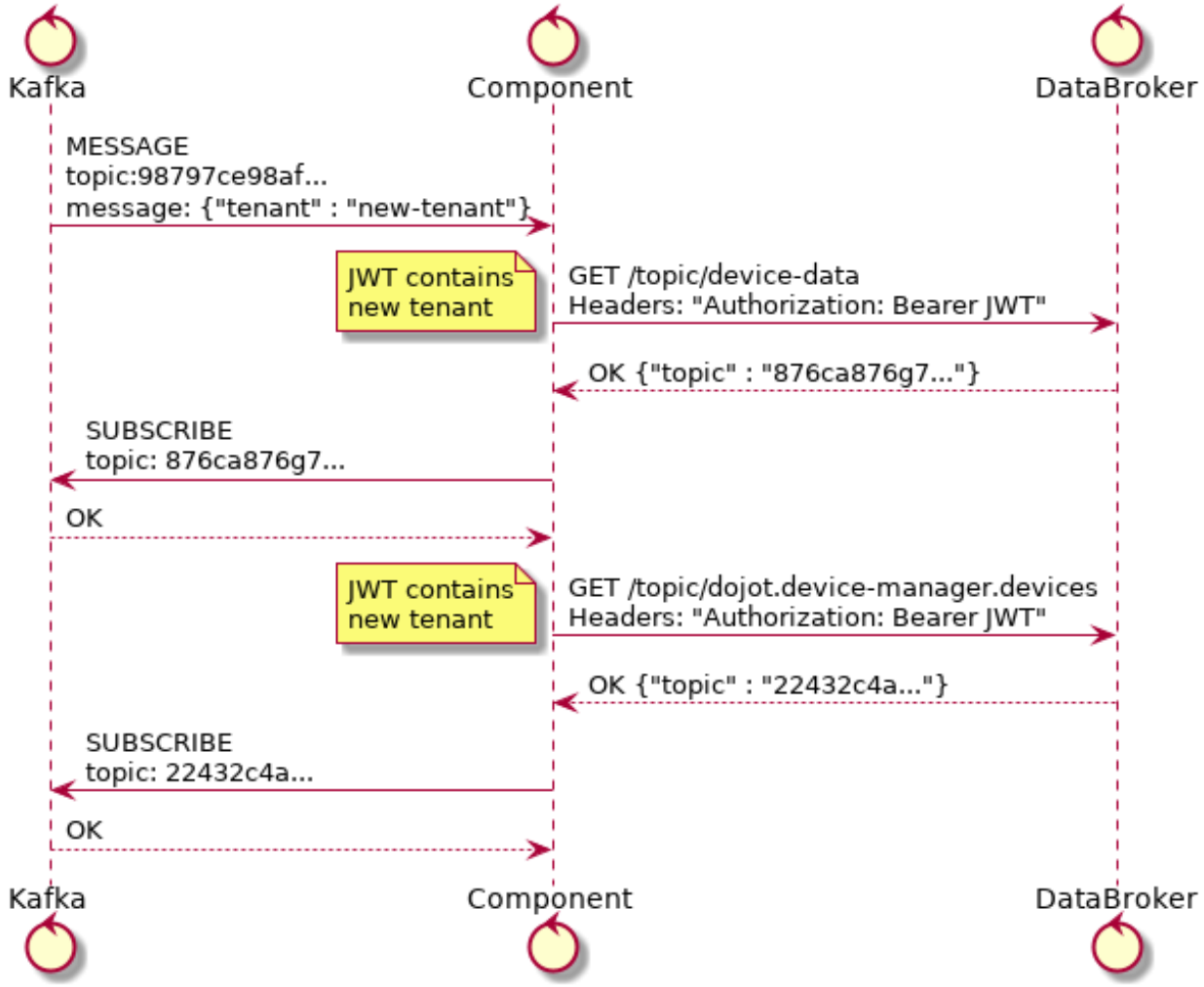


Fig. 5.8: Inicialização do *tenant*

```
(curl -o /dev/null ${kong}/apis -sS -X POST \
    --header "Content-Type: application/json" \
    -d @- ) <<PAYLOAD
{
    "name": "data-broker",
    "uris": ["/device/(.*)/latest", "/subscription"],
    "strip_uri": false,
    "upstream_url": "http://data-broker:80"
}
PAYLOAD
```

Este comando registrará o endpoint */dispositivo/*/latest* e */subscription* e todas as solicitações serão encaminhadas para *http//data-broker:80*. Você pode verificar a documentação sobre como adicionar endpoints na Kong's documentation.

Para alguns dos endpoints registrados, o kong.config.sh adicionará dois plug-ins aos endpoints selecionados:

1. Geração JWT. A documentação para este plugin está disponível na Kong JWT plugin page.
2. Configure um plug-in que encaminhará todas as solicitações de para o Auth. invocará Auth para autenticar solicitações. Este plugin está disponível no PEP-Kong repository.

A solicitação a seguir instala esses dois plug-ins na API do data-broker:

```
curl -o /dev/null -sS -X POST ${kong}/apis/data-broker/plugins -d "name=jwt"
curl -o /dev/null -sS -X POST ${kong}/apis/data-broker/plugins -d "name=pepkong" -d
↪"config.pdpUrl=http://auth:5000/pdp"
```

Mensagens emitidas

O Auth emitirá apenas uma mensagem via Kafka para a criação do tenant:

```
{
  "type" : "CREATE",
  "tenant" : "XYZ"
}
```

5.4 Device Manager

O DeviceManager armazena e recupera modelos de informações para dispositivos e modelos e algumas informações estáticas sobre eles também. Sempre que um dispositivo é criado, removido ou apenas editado, ele publica uma mensagem no Kafka. Depende apenas do DataBroker e Kafka pelos motivos já explicados neste documento.

Todas as mensagens publicadas pelo Device Manager no Kafka podem ser vistas no Device Manager Device Manager mensagens.

5.5 Agente IoT

Os agentes de IoT recebem mensagens de dispositivos e os convertem em uma mensagem padrão a ser publicada em outros componentes. Para fazer isso, eles podem querer saber quais dispositivos são criados para filtrar corretamente as mensagens que não são permitidas na dojot (usando, por exemplo, informações de segurança para bloquear mensagens de dispositivos não autorizados). Ele usará o assunto `device-data` e a inicialização de *tenants*, conforme descrito em *Inicialização dos tenants*.

Após solicitar os tópicos para todos os *tenants* no assunto *device-data*, o agente da IoT começará a receber dados dos dispositivos. Como há várias maneiras pelas quais os dispositivos podem fazer isso, esta etapa não será detalhada nesta seção (isso depende muito de como cada agente de IoT funciona). No entanto, ele deve enviar uma mensagem para Kafka para informar outros componentes de todos os novos dados que o dispositivo acabou de enviar. Isso é mostrado na Fig. 5.9

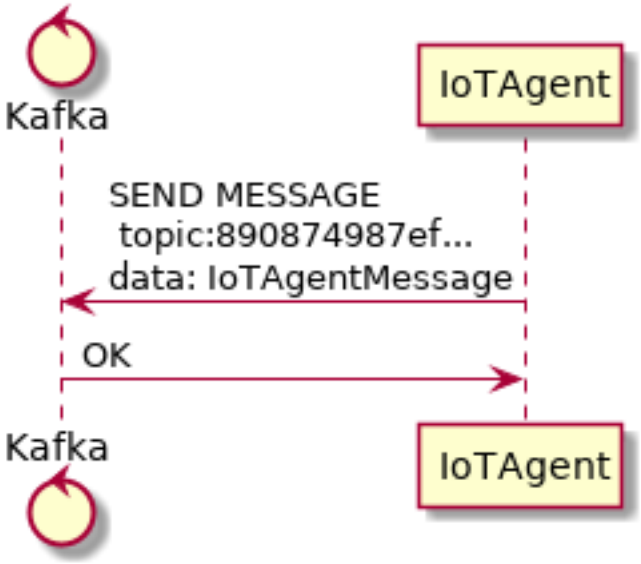


Fig. 5.9: Mensagem do agente de IoT para Kafka

Os dados enviados pelo agente de IoT têm a estrutura mostrada na Fig. 5.10.

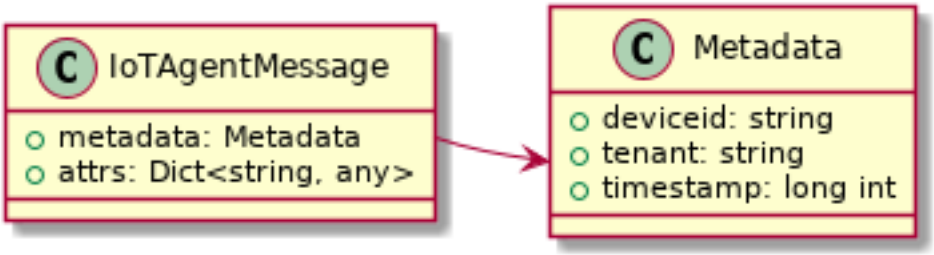


Fig. 5.10: Estrutura de mensagens do agente IoT

Essa mensagem seria:

```
{
    "metadata": {
        "deviceid": "c6ea4b",
        "tenant": "admin",
        "timestamp": 1528226137452
    },
    "attrs": {
        "humidity": 60,
        "temperature" : 23
    }
}
```

5.6 Persister

Persister é um serviço muito simples, cujo único objetivo é receber mensagens dos dispositivos (usando o assunto `device-data`) e armazená-las no MongoDB. Para isso, é realizado o procedimento de inicialização (detalhado em *Inicialização dos tenants*) e, sempre que uma nova mensagem é recebida, ele cria um novo documento Mongo e o armazena na coleção do dispositivo. Isso é mostrado na Fig. 5.11.

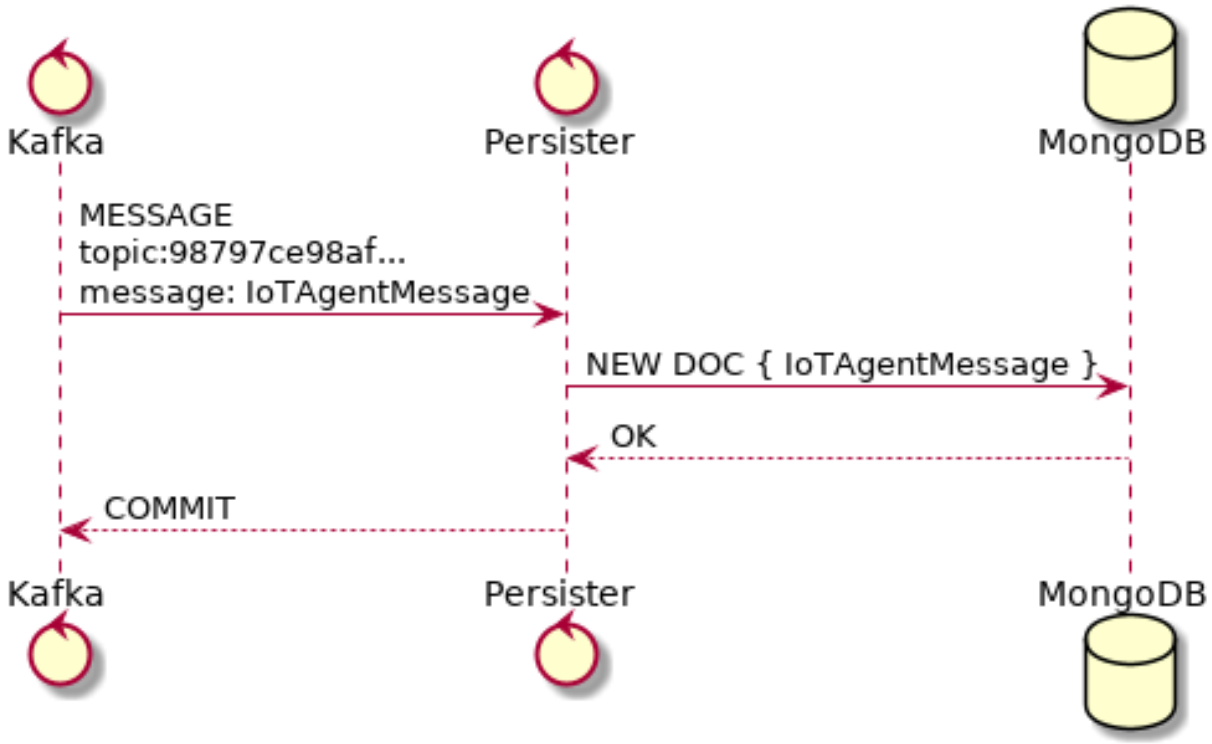


Fig. 5.11: Persister

Este serviço é simples, pois é por design.

5.7 History

O histórico também é um serviço muito simples: sempre que um usuário ou aplicativo envia uma solicitação, ele consulta o MongoDB e cria uma mensagem adequada para enviar de volta ao usuário/aplicativo. Isso é mostrado na Fig. 5.12.

5.8 Data Broker

O DataBroker possui algumas funcionalidades a mais do que apenas gerar tópicos para pares `{tenant, subject}`. Ele também servirá conexões socket.io para emitir mensagens em tempo real. Para fazer isso, ele recupera todos os tópicos para o assunto *device-data*, assim como em qualquer outro componente interessado nos dados recebidos dos dispositivos. Assim que receber uma mensagem, ela será encaminhada para uma 'sala' (usando o vocabulário socket.io) associada ao dispositivo e ao *tenant* associado. Portanto, todo cliente conectado a ele (como interfaces gráficas de usuário) receberá uma nova mensagem contendo todos os dados recebidos. Para obter mais informações sobre como abrir uma conexão socket.io com o DataBroker, consulte a Documentação do DataBroker.

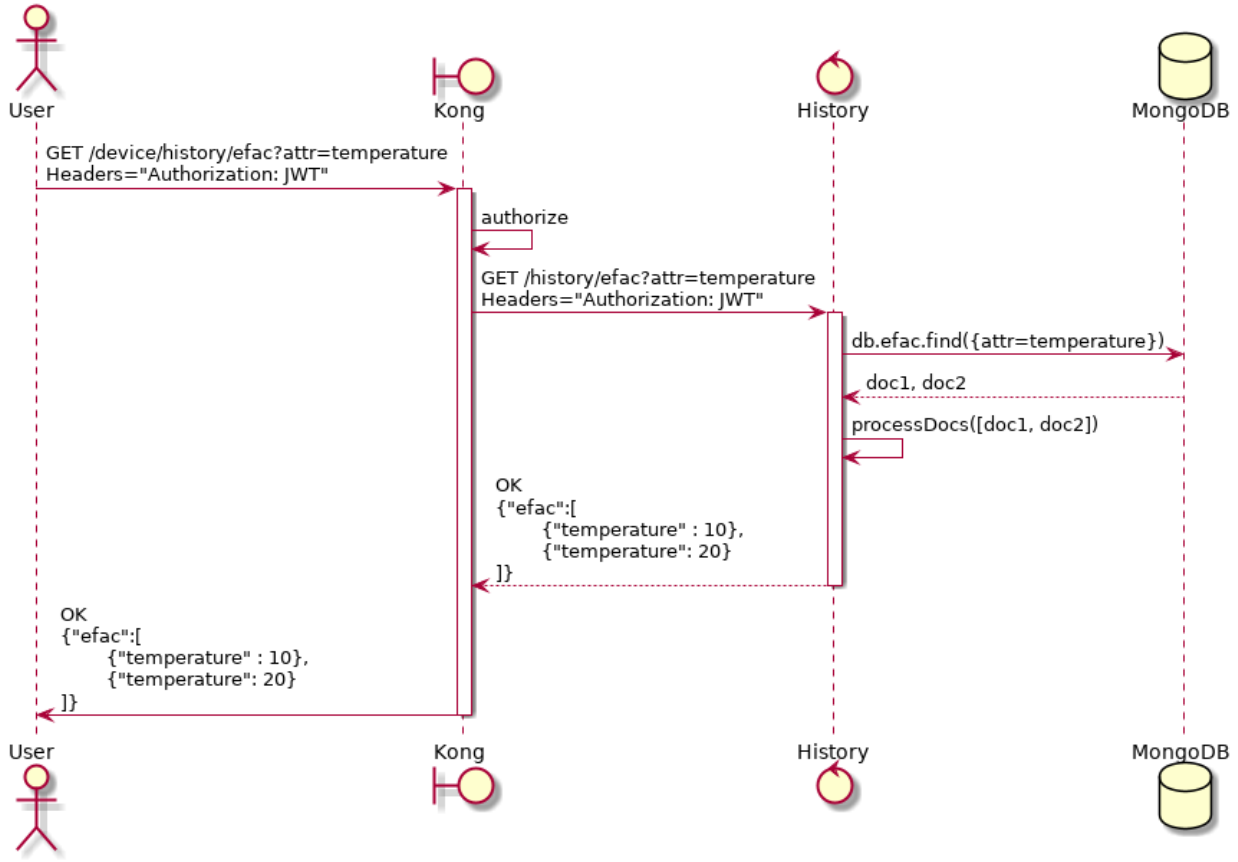


Fig. 5.12: History

CAPÍTULO 6

Guia de instalação

Esta página contém informação de como instalar a dojot utilizando o docker-compose e o Kubernetes.

Table of Contents

6.1 Requisitos de hardware

Os requisitos de hardware estimados para 500 dispositivos com atualizações a cada 15s são:

Tabela 6.1: Requisitos de hardware para 500 dispositivos

Implantação dojot		CPU	RAM	Espaço livre em disco
Docker-compose		4 Núcleos	4GB	10GB
Kubernetes	Master	2 Núcleos	2GB	2GB
Kubernetes	Worker	4 Núcleos	4GB	10GB

Além disso, são necessários:

- **Acesso à rede**
- **As seguintes portas devem estar abertas:**
  - **TCP**: 8000 *(web interface access)*; 1883 *(MQTT, If you are going to use MQTT)*; 5896 *(LW2M, If you are going to use LW2M file server via HTTP instead of coap, UDP).*
  - **TLS**: 8883 *(MQTTS, If you are going to use MQTT with TLS, in secure mode.).*
  - **UDP**: 5683 and 5693 *(LW2M, If you are going to use LW2M)*; 5684 and 5694 *(LW2M, If you are going to use LW2M with DTLS).*

Nota: Os núcles acima são de aproximadamente 3.5 GHz (x86-64)

6.2 Docker-compose

Neste video tutorial acima é utilizada a versão v0.4.2, mas o mesmo video é válido para a versão atual, apenas é necessário mudar pra a versão v0.4.3

Este documento contém instruções de como criar um ambiente para instalação trivial da dojot em um único host utilizando o docker-compose como o processo de orquestração da plataforma.

Muito simples, esta opção de instalação é a que melhor se adapta para desenvolvimento e verificação da plataforma dojot, mas não é aconselhável para ambientes de produção.

Este guia foi verificado utilizando-se o sistema operacional Ubuntu 16.04 e 18.04 LTS.

As seções seguintes descrevem todas as dependências do docker-compose.

6.2.1 Docker engine (motor do docker)

Informações atualizadas e procedimentos de instalação para o docker engine podem ser encontrados na documentação do projeto:

https://docs.docker.com/engine/install/ubuntu/

Nota: Um passo adicional no processo de instalação e configuração do docker em um determinado equipamento é definir quem será elegível para criar/iniciar instâncias do docker.

Caso os passos pós-instalação não tiverem sido executados (mais especificamente o "Manage docker como usuário não-root"), todos os comandos do docker e do docker-compose devem ser executados pelo super usuário (root), ou invocando o sudo.

https://docs.docker.com/engine/installation/linux/linux-postinstall/

6.2.2 Docker-compose

Informações atualizadas sobre procedimentos de instalação para o docker-compose podem ser encontradas na documentação do projeto:

https://docs.docker.com/compose/install/

6.2.3 Instalação

Para construir o ambiente, simplesmente clone o repositório e execute os comandos abaixo.

O repositório com os scripts de instalação e configuração do docker-compose podem ser encontrados em:

https://github.com/dojot/docker-compose

ou com o comando git clone:

```
git clone https://github.com/dojot/docker-compose.git
# Let's move into the repo - all commands in this page should be executed
# inside it.
cd docker-compose
```

Uma vez que o repositório esteja propriamente clonado, selecione a versão a ser utilizada por meio da tag apropriada (note que o tagname deve ser substituído):

```
# Must be run from within the deployment repo

git checkout tag_name -b branch_name
```

Por exemplo:

```
git checkout v0.4.3 -b v0.4.3
```

Feito isso, o ambiente pode ser iniciado assim:

```
# Must be run from the root of the deployment repo.
# May need sudo to work: sudo docker-compose up -d
docker-compose up -d
```

Para verfificar o estado de um container individual, comandos do docker podem ser utilizados, como por exemplo:

```
# Shows the list of currently running containers, along with individual info
docker ps

# Shows the list of all configured containers, along with individual info
docker ps -a
```

Nota: Todos os comandos para docker e docker-compose podem requerer credenciais de super usuário (root) ou sudo.

Para permitir usuários "não-root" gerenciar o docker, confira a documentação do docker:

https://docs.docker.com/engine/installation/linux/linux-postinstall/

6.2.4 Utilização

A interface web está disponível em `http://localhost:8000`. O usuário é `admin` e a senha é `admin`. Você também pode interagir com a plataforma utilizando o Components and APIs.

Leia o *Utilizando a API da dojot* e *Usando a interface WEB* para maiores informações sobre como interagir com a plataforma dojot.

6.3 Kubernetes

Neste video tutorial acima é utilizada a versão v0.4.2, mas o mesmo video é válido para a versão atual, apenas é necessário mudar pra a versão v0.4.3

This section provides instructions on how to create a dojot deployment on a multi-node environment, using Kubernetes as the orchestration platform.

Esta opção de implantação, quando configurada corretamente, pode ser usada para criar ambientes de produção.

As seções a seguir descrevem todas as dependências e etapas necessárias para esta implantação.

6.3.1 Cluster Kubernetes

Para este guia, é recomendável que você já tenha um cluster K8s funcionando.

Se você precisar criar um cluster Kubernetes do zero, informações atualizadas e procedimentos de instalação podem ser encontrados na documentação de instalação do Kubernetes.

6.3.2 Kubernetes Requisitos

- A versão suportada pelo Kubernetes é entre **v1.11** e **v1.15**.
- Acesso aos repositórios do Docker Hub
- (opcional) uma classe de armazenamento que será usada para armazenamento persistente

6.3.3 Implantação dojot

Para implantar o dojot no Kubernetes, é recomendável o uso de *playbooks ansible* desenvolvidos para o dojot. Os manuais e todo o código relacionado podem ser encontrados no repositório *Ansible* dojot.

As etapas a seguir descreverão como usar este repositório e o *ansible*.

1. Clonando o repositório

A primeira etapa da implantação é a clonagem do repositório. Para fazer isso, execute o comando:

```
git clone -b v0.4.3 https://github.com/dojot/ansible-dojot
```

2. Instalando dependências

A próxima etapa é instalar as dependências para executar o *ansible*, essas dependências incluem o próprio *ansible* com outros módulos que serão usados para analisar modelos e se comunicar com os kubernetes.

Digite a pasta em que o repositório foi baixado e instale os pacotes pip com os seguintes comandos:

```
cd ansible-dojot
pip install -r requirements.txt
```

3. Configurando o inventário

Para implantar o kubernetes com *ansible*, é necessário modelar o ambiente desejado em um inventário *ansible*.

No repositório, há uma pasta 'inventory' que contém um exemplo de inventário chamado 'example_local' que pode ser usado como ponto de partida para criar o inventário do ambiente real.

O primeiro arquivo que requer alterações é o hosts.yaml. Este arquivo descreve os nós que serão acessados pelo *ansible* para executar a implantação. Como a implantação do dojot é feita diretamente nos K8s, apenas um nó com acesso ao cluster kubernetes é realmente necessário.

O nó que acessará o cluster pode ser um nó de cluster kubernetes acessível via SSH ou evento sua máquina local se puder alcançar o cluster kubernetes com um arquivo de configuração.

No arquivo de exemplo, o acesso é feito através de um nó local, onde o *script ansible* é executado. Este nó é descrito como localhost no item de hosts do grupo **all**.

Esses mesmos nós devem ser adicionados como filhos do grupo dojot-k8s.

Para configurar um acesso local no arquivo hosts, siga o exemplo abaixo:

```
---
all:
  hosts:
    localhost:
      ansible_connection: local
      ansible_python.version.major: 3
  children:
    dojot-k8s:
      hosts:
        localhost:
```

Para configurar o acesso remoto via ssh para um nó do cluster, siga este outro exemplo:

```
---
all:
  hosts:
    NODE_NAME:
      ansible_host: NODE_IP
  children:
    dojot-k8s:
      hosts:
        NODE_NAME:
```

A próxima etapa é configurar as variáveis obrigatórias e opcionais necessárias para implementar a dojot.

Há um documento descrevendo cada uma das variáveis que podem ser configuradas nas variáveis *Ansible* dojot.

Essas variáveis devem ser definidas para o grupo *'dojot-k8s*', para isso, defina seus valores no arquivo dojot.yaml na pasta '**group_vars/dojot-k8s/**'

4. Executando a implantação

Agora que o inventário está definido, a próxima etapa é executar o *playbook* de implantação

Para fazer isso, execute o seguinte comando:

```
ansible-playbook -K -k -i inventories/YOUR_INVENTORY deploy.yaml
```

Aguarde a execução do *playbook* terminar sem erros.

5. Acessando o ambiente dojot implantado

O acesso a Dojot será definido usando NodePorts. Para visualizar as portas apropriadas para acessar o ambiente, é necessário verificar a configuração do serviço.

```
kubectl get service -n dojot kong iotagent-mosca
```

Este comando retornará a porta usada para acesso externo à API REST e à GUI via kong e à porta MQTT via iotagent-mosca.

CAPÍTULO 7

Dúvidas Mais Frequentes

Aqui estão algumas respostas às dúvidas mais frequentes sobre a plataforma dojot.

Não encontrou aqui uma resposta para a sua dúvida? Por favor, abra uma *issue* no repositório da dojot no Github.

Sumário

7.1 Gerais

7.1.1 O que é a dojot? Por que eu deveria utilizá-la? Por que abrir o código?

É uma plataforma brasileira para IoT que surgiu com uma proposta de código aberto, para facilitar o desenvolvimento de soluções e o ecossistema IoT com conteúdo local voltado às necessidades brasileiras.

Ela oferece

- APIs abertas tornando o acesso fácil das aplicações aos recursos da plataforma.
- Capacidade de armazenamento de grandes volumes de dados em diferentes formatos.
- Conectores para diferentes tipos de dispositivos.
- Construção de fluxos de dados e regras de forma visual, permitindo a rápida prototipação e validação de cenários de aplicações IoT.

- Processamento de eventos em tempo real aplicando regras definidas pelo desenvolvedor.

7.1.2 Onde eu posso baixar?

Todos os componentes estão disponíveis no repositório da dojot no GitHub: https://github.com/dojot.

7.1.3 Qual é o principal repositório?

Existem 3 repositórios principais:

- https://github.com/dojot/dojot: é aqui que concentramos o acompanhamento de tudo relacionado a este projeto como decisões de arquitetura e melhorias.
- https://github.com/dojot/docker-compose: repositório com os arquivos e configurações para o docker-compose. Este é o repositório que recomendamos para começar com a dojot.
- https://github.com/dojot/ansible-dojot: repositório com os arquivos e configurações para utilizar kubernetes.

7.1.4 Então, encontrei um probleminha chato. Como posso informá-lo sobre isso?

Não encontrou aqui uma resposta para a sua dúvida? Por favor, abra uma *issue* no repositório da dojot no Github.

Se você puder analisar e resolver o problema, por favor faça isso e crie um *pull-request* com uma breve descrição do que foi feito.

7.2 Uso

7.2.1 Por onde eu começo? É baseado em CLI ou possui uma interface gráfica de usuário ?

A dojot pode ser acessada via interface Web ou APIs REST. Considerando que você já tenha instalado o `docker` e o `docker-compose` e tenha clonado o repositório dojot para o `docker-compose`, para iniciar todos os serviços, basta executar o comando abaixo:

```
$ docker-compose up -d
```

E é isto.

A interface Web está disponível em `http://localhost:8000`. O usuário é `admin` e a senha é `admin`.

APIs REST são explicadas na seção *Aplicações*.

7.2.2 Pronto, já iniciei e fiz o *login*. E agora?

Legal! Agora você pode criar seus primeiros templates e dispositivos, descrito em *Dispositivos*, criar alguns fluxos e registrar-se para eventos de dispositivos, ambos descritos em *Fluxos de Dados*.

7.2.3 Como posso atualizar o meu ambiente com a última versão da dojot?

Basta seguir alguns passos:

1 Atualize o repositório do docker compose com a última versão.

```
$ cd <path-to-your-clone-of-docker-compose>
$ git checkout master && git pull
```

Se você precisar de uma outra versão, você pode fazer *checkout* de uma tag:

```
$ git tag
  0.1.0-dojot
  0.1.0-dojot-RC1
  0.1.0-dojot-RC2
  0.2.0
  v0.3.0-beta.1
  v0.3.1
  v0.4.0
  v0.4.1
  v0.4.1_rc2
  v0.4.2
  v0.4.2-rc.1
  v0.4.3-rc.1
  v0.4.3-rc.2
  v0.4.3


$ git checkout v0.4.2
```

De tempos em tempos nós lançaremos novas versões para componentes dadojot. Eles podem ser lançados individualmente (no entanto, tentaremossincronizar todos os lançamentos). Uma vez que nós tenhamos um conjunto decomponentes que opere de forma estável, atualizaremos o repositório do docker-compose

Atualize o ambiente com as imagens dockers mais recentes.

```
$ docker-compose pull && docker-compose up -d
```

Este procedimento também se aplica para as máquinas virtuais dojot uma vez que as mesmas utilizam *docker-compose*.

7.3 Dispositivos

7.3.1 O que são dispositivos para a dojot?

Na dojot, um dispositivo é uma representação digital para um dispositivo real ou gateway com um ou mais sensores ou uma representação para um dispositivo virtual com sensores/atributos inferidos de outros dispositivos.

Considere, por exemplo, um dispositivo real com um termômetro e umhigrômetro; ele pode ser representado na dojot como um dispositivo com doisatributos (um para cada sensor). Chamamos este tipo de dispositivo de*dispositivo normal* ou usando o seu protocolo de comunicação, como*dispositivo MQTT* ou *dispositivo CoAP*.

Nós também podemos criar dispositivos que não correspondem diretamente a dispositivos reais, por exemplo, podemos criar um dispositivo com informação em alto nível de temperatura (*está ficando mais quente* ou *está ficando mais frio*) cujos valores são inferidos a partir de sensores de temperatura de outros dispositivos. Este tipo de dispositivo é denominado de *dispositivo virtual*.

7.3.2 Qual é a relação entre este *dispositivo* e um dispositivo real?

É simples como parece: o *dispositivo* para a dojot é um espelho (gêmeo/cópia digital) do dispositivo real. Você pode escolher quais atributos são disponibilizados para as aplicações e outros componentes, adicionando cada um deles através da interface de criação de dispositivo.

7.3.3 O que são *dispositivos virtuais*? Como se diferenciam dos demais?

Dispositivos são criados para serem como espelhos (gêmeo/cópia digital) dos dispositivos e sensores reais. Um *dispositivo virtual* é uma abstração que modela coisas que não são factíveis no mundo real. Por exemplo, digamos que um usuário tenha alguns sensores para detectar fumaça em um laboratório, sendo que cada um tem diferentes atributos.

Não seria bom se exisitisse um dispositivo chamado *Laboratório* quepossui um atributo *emChamas*? Assim a aplicação dependeria apenas desteatributo para tomar alguma ação.

Uma outra diferença é a maneira como os dados dos dispositivos virtuais são populados. Os dispositivos são preenchidos com informações enviadas aplataforma dojot por dispositivos ou gateways e os virtuais são preenchidos por fluxos ou por aplicações.

7.3.4 E o que são *templates*?

Templates são "modelos para dispositivos" que servem como base para acriação de um novo dispositivo. Um dispositivo é construído usando um conjuntode templates - seus atributos serão herdados de cada template (não deve havernenhum atributo com mesmo nome, no entanto). Se um template é alterado, todosos dispositivos associados àquele template serão também alteradosautomaticamente.

7.3.5 Como posso enviar dados via MQTT para a dojot de forma que apareçam no *dashboard*?

Primeiramente, crie uma representação digital para o seu dispositivo real. Depois, configure o seu dispositivo real para enviar dados para a dojot de maneira que os dados possam ser associados ao seu representante digital.

Tomemos como exemplo uma estação meteorólogica que monitora temperatura e umidade e publica essas medidas via MQTT periodicamente. Inicialmente, cria-se um dispositivo do tipo MQTT com dois atributos (temperatura e umidade) e, em seguida, configura-se a estação meteorológica para publicar os dados para a dojot.

Para enviar dados para a dojot via MQTT (usando o iotagent-mosca),existem algumas coisas para se ter em mente:

- O tópico deve parecer com `/<tenant>/<device-id>/attrs` (por exemplo: `/admin/efac/attrs`). Dependendo de como o IoT agent foi inicializado (mais restritivo), o `client ID` deve ser também configurado para''<tenant>:<deviceid>'', como `"admin:efac"`.
- O *payload* MQTT precisa ser um JSON com as chaves correspondendo aos atributos definidos para o dispositivo na dojot, como:

```
{ "temperature" : 10.5,"pressure" : 770 }
```

7.3.6 No *dashboard* alguns atributos são exibidos como tabelas e outros como gráficos. Como são escolhidos/configurados?

O tipo do atributo determina o modo de exibição dos dados no *dashboard* como segue:

- `Geo`: mapa georreferenciado.

- `Boolean` e `Text`: tabela
- `Integer` e `Float`: gráfico de linha.

7.3.7 Estou interessado em integrar à dojot o meu dispositivo que é super legal. Como eu faço isso?

Se o seu dispositivo envia mensagens via MQTT (com *payload* do tipo JSON), CoAP ou HTTP, existe uma grande chance de ser possível intergrá-lo com mínima ou nenhuma modificação.

7.3.8 Existem restrições para as mensagens enviadas pelo meu dispositivo para a dojot? Formato, tamanho, frequência?

Nenhuma com exceção do formato, que é descrito na questão *How can I send MQTT data to dojot so that it appears on the dashboard?*.

7.3.9 Como posso enviar comandos para o meu dispositivo através da dojot?

Por enquanto, você pode enviar requisições HTTP para a dojot contendoalgumas instruções sobre qual dispositivo deve ser configurado e os dados paraatuação. Mais detalhes podem ser encontrados na seção Device-Manager how-to - sending actuation messages.

7.3.10 Não encontrei o protocolo suportado pelo meu dispositivo na lista de tipos, existe algo que eu possa fazer?

Existem algumas possibilidades. A primeira é desenvolver um *proxy* para traduzir o seu protocolo para um dos suportados pela dojot. A segunda é desenvolver um IotAgent, conector, similar as existentes para MQTT, CoAP e HTTP. Dê uma olhada em https://github.com/dojot/iotagent-nodejs

7.3.11 Eu salvei um atributo, mas o mesmo sumiu do dispositivo. É um defeito?

Provavelmente você salvou o atributo, mas não o dispositivo. Se você não clicar no botão para salvar o dispositivo, o atributo adicionado será descartado. Estamos melhorando as mensagens da plataforma para avisar e lembrar os usuários de salvarem as suas configurações.

7.3.12 Compo eu posso obter dados históricos para um dispositivo em particular?

Você pode fazer isto enviando uma requisição para */history/*, como:

```
curl -X GET \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsIn...' \
  "http://localhost:8000/history/device/3bb9/history?lastN=3&attr=temperature"
```

o qual retornará as últimas 3 entradas do atributo *temperature* dodispositivo *3bb9*

```
[
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:47:07.050000Z",
```

(continues on next page)

(continuação da página anterior)

```
    "value": 29.76,
    "attr": "temperature"
  },
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:46:42.455000Z",
    "value": 23.76,
    "attr": "temperature"
  },
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:46:21.535000Z",
    "value": 25.76,
    "attr": "temperature"
  }
]
```

Há mais operadores que podem ser usados para filtrar entradas. Check History API para ver todas os possíveis operadores e outros filtros.

7.4 Fluxos de Dados

7.4.1 O que é um fluxo?

É um processamento de mensagens de um dispositivo. Com um fluxo, você pode analisar dinamicamente cada nova mensagem para fazer validações, inferirinformações e tomar ações ou gerar notificações.

7.4.2 A interface dos fluxos... ela se parece com o node-RED. Eles tem alguma relação?

A interface dos fluxos é baseada no node-RED, mas a aplicação das regras e execução das ações é feita por um mecanismo próprio da dojot. Se o node-REDfor familiar para você, não será difícil usar o flowbroker.

7.4.3 Por que eu deveria usar um fluxo?

Ele permite uma das coisas mais interessantes do IoT de uma forma simples e intuitiva que é analisar dados para extração de informações e execução de ações.

7.4.4 O que ele pode fazer, exatamente?

Você pode fazer coisas como:

- Criar visões para um dispositivo (renomear atributos, agregá-los, alterá-los, etc.)
- Inferir informações baseadas em regras de detecção de borda e georreferenciamento.
- Enviar notificações via email.
- Enviar notificações via HTTP.

O componente responsável pelo fluxo de dados está em desenvolvimento constante e novas funcionalidades são adicionadas a cada versão.

Há mecanismos para adicionar novos blocos de processamento a fluxos. Veja *How can I add a new node type to its menu?* para mais informações sobreisto.

7.4.5 Pois bem, como eu posso usá-lo?

Ele segue o modo de uso do node-RED. Você pode ler a documentação para mais detalhes.

7.4.6 Posso aplicar o mesmo fluxo para múltiplos dispositivos?

Você pode usar um template como entrada para indicar que ele deve serexecutado para todos os dispositivos associados àquele template. É válidoindicar que o fluxo é sempre executado para cada mensagem.

7.4.7 Posso correlacionar dados de diferentes dispositivos no mexmo fluxo?

Uma vez que os fluxos são aplicados individualmente para cada mensagem, você deve criar um dispositivo virtual para agregar todos os atributos e então usar este dispositivo como entrada de um novo fluxo.

Outra coisa que pode ser feita é construir um nó do flowbroker paralidar com contextos, os quais podem ser usados para armazenar e obter dadosrelacionados a um fluxo ou nó.

7.4.8 E se eu quiser enviar um HTTP POST?

Um aviso importante: assegure-se de que a dojot consegue acessar seu servidor.

7.4.9 Eu quero renomear os atributos de um dispositivo. O que eu devo fazer?

Primeiramente, você deve criar um dispositivo virtual com os novos atributos, para então construir um fluxo de dados para renomeá-los. Isto pode ser feito conectando um nó 'change' após um dispositivo de entrada para mapear os atributos de entada a seus correspondentes na saída.

7.4.10 Quero agregar os atributos de múltiplos dispositivos. O que eu devo fazer?

Inicialmente, você deve criar um dispositivo virtual para agregar todos os atributos. Com este dispositivo criado, você deve criar fluxos para mapeamento dos atributos de cada dispositivo real de entrada neste dispositivo virtual. Isto pode ser feito em nós 'change' conectados a cada um dos dispositivos de entrada a fim de criar uma variável contendo todos os atributos de saída. Todos os nós change devem ser, por fim, conectados ao nó de saída representando o dispositivo virtual.

7.4.11 Como eu posso adicionar um novo tipo de nó no menu?

E é isso! Em https://github.com/dojot/flowbroker/tree/master/lib há 2 exemplos de como construir uma imagem do Docker que pode ser adicionada ao menu de nós do flowbroker.

7.5 Aplicações

7.5.1 Quais APIs estão disponíveis para aplicações?

Você pode ver todas as APIs disponíveis na página API Listing page

7.5.2 Como posso usá-los?

Há um tutorial simples e rápido na in the Using API interface.

7.5.3 Estou interessado(a) em integrar minha aplicação com dojot. O que devo fazer?

Isto deve ser bastante direto. Há duas formas de integrar sua aplicação à dojot:

- **Obtenção de dados históricos**: você pode querer ler todos os dados históricos relacionados a um dispositivo de forma periódica. Isto pode se feito usando esta API (um lembrete apenas: todos os serviços descritos neste apiary deve ser precedido de `/history/`).
- **Usar os fluxos de dados para pré-processar dados**: se for necessário realizar algum processamento extra, você pode usar os fluxos. Eles auxiliam no processamento e na transformação de dados para envio para sua aplicação via requisições HTTP. Uma outra forma é armazenar os dados em dispositivos virtuais e criar subscrições para enviar notificações toda vez que acontecer uma alteração em seus atributos.

Todas as requisições devem carregar um token de acesso, o qual pode ser obtido como descrito na pergunta *Como posso usá-los?*.

CAPÍTULO 8

Histórico de lançamento

8.1 carate - 2019.09.11

- Agentes IoT:
  - Suporte para dispositivos LWM2M
- GUI:
  - Nova interface para gerenciamento de dispositivos
  - Nova interface para gerenciamento de modelos de dispositivos
  - Gerenciamento de metadados de atributo
  - Importar e exportar
  - Atualização de Firmware
  - Perfis
  - Notificações
  - Internacionalização
  - Exibir detalhes de atributos de atuação
  - Filtro de dispositivos no mapa
- Flows:
  - Novo Nó Evento Dispositivo (Entrada)
  - New node Event Template Device
  - Novo nó FTP
  - Novo nó de Saída para vários dispositivos (Multi Device Out)
  - Novo nó de Notificações
  - Novo nó de atuação para vários dispositivos (Multi Actuate)

  - Novo nó Cron
  - Novo nó Cron Batch
  - Novo nó de Soma acumulada
  - Novo nó de Mesclar dados
  - Nó de email foi removido
  - Internacionalização
  - Suporte a *handlebars template* no nó de template
- History:
  - Novo endpoint para consultar notificações
- ImageManager:
  - Melhorias para dar suporte à atualização de firmware
- DataBroker:
  - Suporte a Notificações no socket-io
- DataManager:
  - É um novo microsserviço da dojot que gerencia a configuração de dados da dojot, tornando possível importar e exportar as configurações.
- Cron:
  - É um novo microsserviço da dojot que permite agendar eventos a serem emitidos para outros microsserviços.
- Novas bibliotecas:
  - Para acelerar o desenvolvimento: Dojot Module Java e Dojot Module Python
  - HealthCheck: HealthCheck Python e Healthcheck NodeJs
  - IoTAgent: IoTAgent Java

CAPÍTULO 9

Usando a interface WEB

Este tutorial descreve as operações básicas na dojot, como criar dispositivos, conferir seus atributos e criar fluxos, importar/exportar e atualizar firmware.

Nota:

- Para quem é esse tutorial: usuários iniciantes
- Nível: básico
- Tempo de leitura: 20 minutos

9.1 Gerenciamento de dispositivo

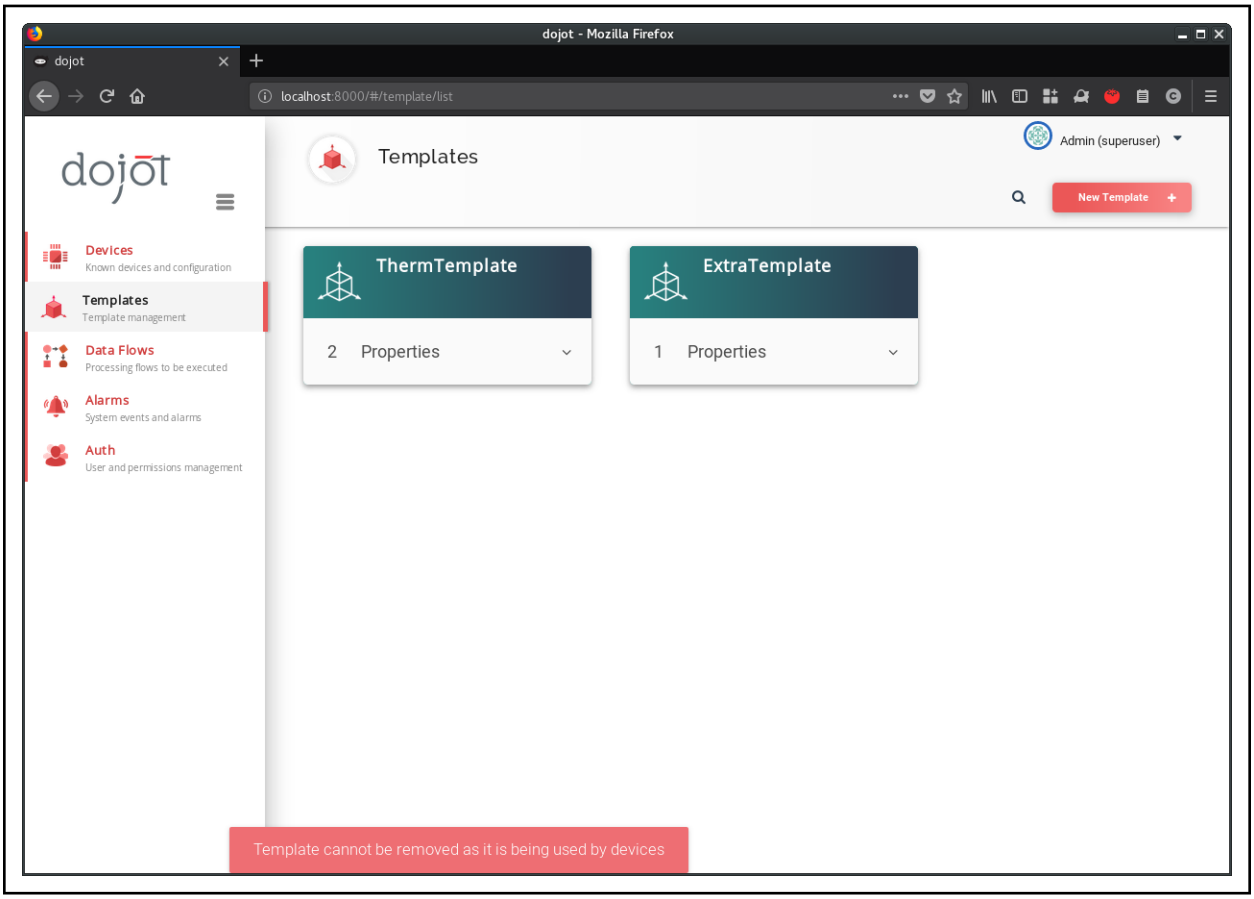
Esta seção mostra como gerenciar dispositivos. Para tal, serão utilizados dois dispositivos sensores de temperatura e um dispositivo virtual, esse último com a função de observar as temperaturas medidas nos dois primeiros e gerar alarmes em determinadas condições.

Como descrito em Concepts, todos os dispositivos são baseados em um ou mais modelos (templates). Para a criação de um modelo, você deve acessar a opção Modelos (Templates) na lateral esquerda da tela e então criar um Novo Modelo (New Template), como mostrado abaixo.

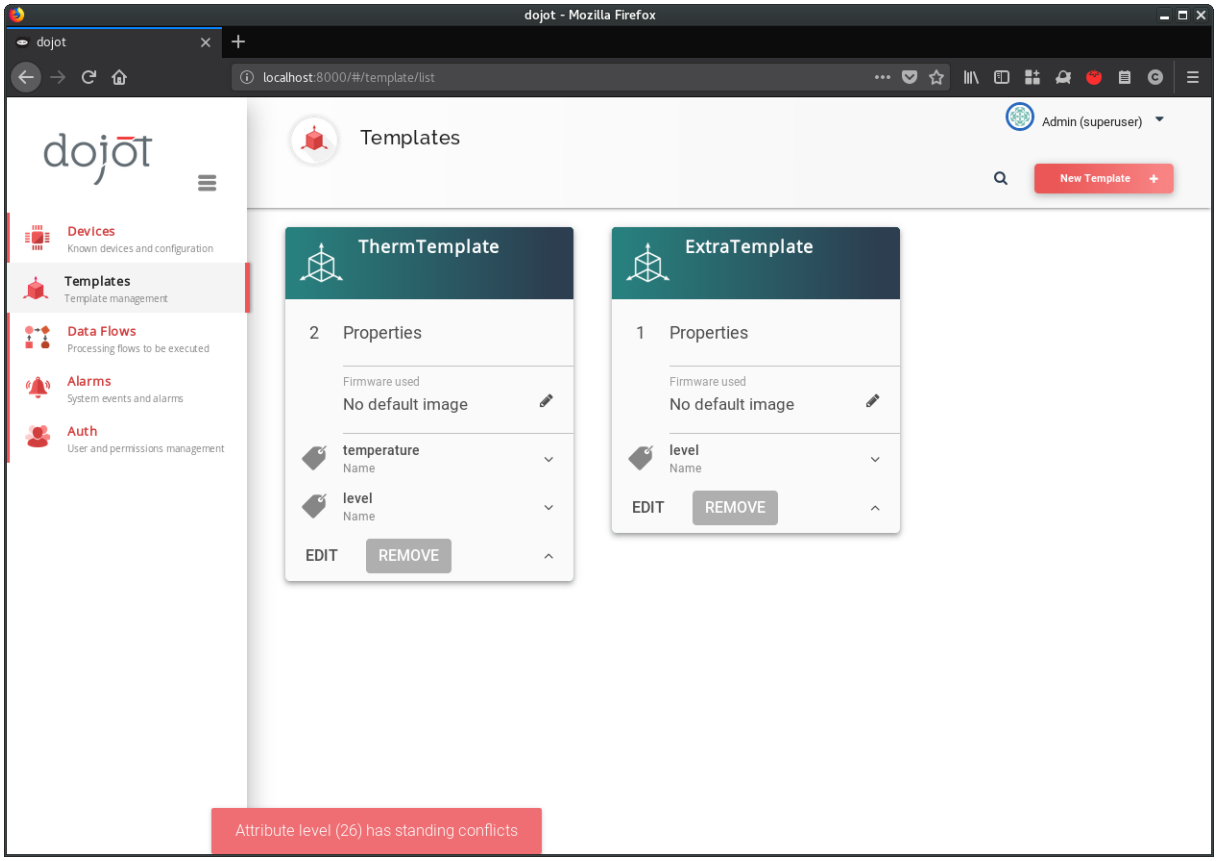
Para criar novos dispositivos, deve-se voltar para a opção Dispositivos (Devices) e criar um Novo Dispositivo (New Device), selecionando os modelos nos quais o dispositivo será baseado, como mostrado abaixo.

Note que, quando um modelo é selecionado no painel direito da tela de criação de dispositivo, todos os atributos são herdados para aquele dispositivo. É possível adicionar mais de um modelo, tendo em mente que modelos que compõem o dispositivo não podem compartilhar atributos com o mesmo nome.

Atenção: Os dispositivos são fortemente atrelados aos a modelos. Para remover um modelo, deve-se remover primeiro todos os dispositivos a ele associados. Caso contrário, a seguinte mensagem aparecerá:



Atenção: É possível adicionar e remover atributos dos modelos, fazendo com que as alterações sejam imediatamente refletidas nos dispositivos associados. No caso de novos adição, no entanto, deve-se observar que os atributos dos modelos que compõem um determinado dispositivo não podem possuir o mesmo nome. Se isso acontecer, a seguinte mensagem aparecerá:



Essa imagem da tela foi capturada quando um novo modelo foi criado (`ExtraTemplate`) com um atributo chamado `level`. Depois um novo dispositivo baseado em ambos os modelos foi criado e um novo atributo também chamado `level` foi adicionado ao modelo `ThermTemplate`.

Quando isso ocorre, nenhuma modificação é aplicada ao modelo (nenhum atributo com nome "level" relativo ao "ThermTemplate" é criado). Contudo, o atributo é mantido no cartão do modelo para que o usuário perceba o que está acontecendo. Se o usuário atualizar a tela, as informações serão revertidas para o estado que estava antes da modificação.

Agora os dispositivos físicos podem enviar mensagens para dojot. Há poucas coisas paraprestar atenção: o tópico MQTT é‘‘/{TENANT}/{DEVICE_ID}/attrs‘‘.

Por questão de simplicidade, será emulado um dispositivo utilizando-se a ferramenta mosquito_pub. O parâmetro `client-id` será configurado utilizando a opção `-i` do mosquito_pub. Veja mais sobre em *Utilizando a API da dojot* no tópico Enviandos mensagens. Além disso estes exemplos estão usando o MQTT sem TLS, recomendamos *Usando MQTT com segurança (TLS)*.

Estando criados os sensores de temperatura, falta agora a criação do dispositivo virtual. Ele será a representação de um alarme de sistema disparado quando algo ruim for detectado pelos sensores. Por exemplo, se os sensores de temperatura estivessem instalados em uma cozinha, a medição de uma temperatura acima de 40ºC poderia indicar que o local estaria em chamas. Essa representação do alarme poderia ter dois atributos: nível de severidade e mensagem textual, para que o usuário pudesse ser informado do acontecimento.

Assim como "dispositivos regulares", dispositivos virtuais também são baseados em modelos. Portanto, um modelo será criado, como mostrado abaixo.

9.2 Configuração de fluxo

Uma vez criado o dispositivo virtual, pode-se adicionar um fluxo para implementar a lógica por detrás da geração de alarmes. A ideia é: se a temperatura medida for menor ou igual a 40ºC, o sistema de alarmes será atualizado com uma notificação de severidade 4 (média) e uma mensagem indicando que a cozinha está OK. Caso a temperatura medida seja maior que os 40ºC, uma notificação de severidade 1 (muito alta) será enviada com a mensagem que a cozinha está em chamas. Isto é feito como mostrado abaixo.

É importante notar que os nós do tipo "change" têm uma referência a uma entidade "output". Isso pode ser visto como uma simples estrutura de dados, onde existem os atributos `message` e `severity` que casam com aqueles do dispositivo virtual. Este "objeto" é referenciado no nó de saída (output) como uma fonte de dados para o dispositivo que será atualizado (nessa caso, o dispositivo virtual criado). Em outras palavras, pode-se dizer que há uma informação que é transferida dos nós do tipo "change" para o "dispositivo virtual" com os nomes "msg.output.message" e "msg.output.severity", onde "message" e "severity" são atributos do dispositivo virtual.

Vamos, agora, enviar mais algumas mensagens e ver o que acontece para aquele dispositivo virtual.

9.3 Importar e Exportar

Essa seção mostra como usar as funcionalidades Importar e Exportar. Tais opções permitem que seus dados de configuração sejam salvos em um arquivo , no caso de uma exportação, e carregados na dojot, no caso de Importação. Esse arquivo possui o formato *JSON* e contém os dados de modelos, dispositivos, fluxos, nós remotos e tarefas de agendamento que foram cadastrados na sua organização.Para executar o procedimento de exportação dos dados de configurações, expanda o menu no canto superior direito da página, clique em "Import/Export (Import./Exportação)" e então, em "Export (Exportar)", conforme ilustrado no vídeo a seguir:

O arquivo exportado pode ser armazenado como backup e posteriormente importado novamente na Dojot.

Para executar o procedimento de importação de uma configuração, expanda o menu no canto superior direito da página, clique em "Import/Export (Import./Exportação)" e então, em "Import (Importar)". Na janela que aparece é possível arrastar e soltar seu arquivo ou navegar na pasta de destino e selecioná-lo, sendo só permitido adicionar um arquivo de extensão *JSON*, no formato esperado, conforme ilustrado no vídeo a seguir:

> **Atenção:** Ao executar o procedimento de importação todas as configurações atuais da organização tais como: dispositivos, modelos, fluxos, nós remotos e tarefas de agendamento, serão excluídas permanentemente para que as novas sejam criadas. Os dados de histórico não fazem parte do importar e exportar.

9.4 Atualização de Firmware

Durante a vida útil de um dispositivo, pode ser necessário atualizar seu software de controle (firmware) a fim de corrigir eventuais problemas encontrados durante seu uso ou até mesmo adicionar novas funcionalidades. Atualmente a dojot suporta o procedimento de atualização de firmware via o protocolo de comunicação LwM2M. Para saber detalhes a respeito do procedimento para integração com seu dispositivo por favor verifique a especificação do protocolo LwM2M. Caso seu dispositivo se comunica via este protocolo e possui o procedimento de atualização de firmware implementado, você pode seguir os passos a seguir para atualizar a versão do seu dispositivo.

O processo de atualização de firmware é composto por três etapas:

- gerenciamento das imagens
- transferência da imagem para o dispositivo;

- aplicação da imagem no dispositivo

O detalhamento da execução destas encontra-se a seguir.

Para que a habilitação de gerenciamento de firmware seja disponibilizada, é preciso criar um modelo e, depois de salvo, habilitar o gerenciador de firmware. Após isso, é possível enviar as imagens de firmware para o repositório da dojot que são associadas a este modelo. Atenção: a extensão da imagem deve possuir a extensão ".hex".

Note que assim que o Gerenciador de Firmware é habilitado, são atribuídos ao modelo cinco atributos que são usados para suportar a atualização de imagens. Os nomes dos atributos podem ser editados conforme a necessidade da aplicação. Os atributos são:

- Estado do dispositivo (Device State):
  - Estado atual da atualização de firmware
- Resultado da versão de aplicação (Result of apply version)
  - Resultado da última aplicação de atualização de uma imagem de firmware;
- Define qual versão transferir (Sets which version to transfer):
  - Indica ao agente IoT, responsável pelo dispositivo, qual deve ser o nome e a versão da imagem de firmware a ser transferida e atualizada no dispositivo
- Acionar atualização da versão (Trigger version update)
  - Atuador utilizado para iniciar o procedimento de atualização de firmware
- Versão atual da imagem (Current version of the image):
  - Versão atual da imagem de firmware, caso disponibilizado pelo mesmo

Após criar o modelo com a opção de gerenciamento de firmware habilitada, é preciso associá-lo a um dispositivo. Assim é possível transferir uma imagem e aplicá-la no dispositivo, conforme o vídeo abaixo:

Note que, em cada etapa, são exibidos o status e o resultado da aplicação da imagem.

CAPÍTULO 10

Utilizando a API da dojot

Esta seção descreve o passo a passo completo de como criar, alterar, enviar mensagens e conferir dados históricos relativo a um dispositivo. Este tutorial assume que está sendo utilizada a instalacao docker-compose e que todos os componentes necessários estão sendo executados corretamente na dojot.

Nota:

- Audiência: desenvolvedores
- Nível: básico
- Tempo de leitura: 15 minutos

10.1 Pré-requisitos

Vamos utilizar:

- curl para acessar as APIs da plataforma dojot
- jq para processar o retorno JSON das APIs da plataforma dojot.
- mosquitto publicar e se subscrever em iotagent-mosca (or iotagent-mqtt) via MQTT.

Em distribuições Linux baseadas em Debian, você pode executar:

```
sudo apt-get install curl
sudo apt-get install jq
sudo apt-get install mosquitto-clients
```

10.2 Obtendo um token de acesso

Como mencionado em *Autenticação de usuário*, todas as requisições devem conter um token de acesso que seja válido. É possível gerar um novo token enviando a seguinte requisição:

```
JWT=$(curl -s -X POST http://localhost:8000/auth \
-H 'Content-Type:application/json' \
-d '{"username": "admin", "passwd" : "admin"}' | jq -r ".jwt")
```

Checar:

```
echo $JWT
```

Se o intuito for gerar um token para outro usuário, é necessário somente mudar o username e passwd no corpo da requisição. O token ("eyJ0eXAiOiJKV1QiL...") deve ser usado em toda a requisição HTTP enviada para a dojot, colocando-o no cabeçalho da mensagem. A requisição seria algo desse tipo:

```
curl -X GET http://localhost:8000/device \
  -H "Authorization: Bearer eyJ0eXAiOiJKV1QiL..."
```

É importante ressaltar que o token deve estar inteiro no cabeçalho da requisição, não apenas parte dele. No exemplo, somente os primeiros caracteres foram mostrados por questão de simplificação. Todas as demais requisições serão compostas da variável de ambiente chamada `bash ${JWT}` que contém o token obtido da dojot (mais especificamente do componente de autorização da dojot).

10.3 Criação de dispositivo

A fim de configurar um dispositivo físico na dojot, é necessário criar sua representação na plataforma. O exemplo mostrado aqui é apenas uma parte pequena do que é oferecido pelo componente DeviceManager. Para mais informações sobre esse componente, confira o documento DeviceManager how-to.

Primeiramente vamos criar um modelo (template) para o dispositivo, pois todos os dispositivos são baseados em modelos, não esqueça.

```
curl -X POST http://localhost:8000/template \
-H "Authorization: Bearer ${JWT}" \
-H 'Content-Type:application/json' \
-d ' {
  "label": "Thermometer Template",
  "attrs": [
    {
      "label": "temperature",
      "type": "dynamic",
      "value_type": "float"
    },
    {
      "label": "fan",
      "type": "actuator",
      "value_type": "float"
    }
  ]
}'
```

Esta requisição deve retornar a seguinte mensagem:

```
{
  "result": "ok",
  "template": {
    "created": "2018-01-25T12:30:42.164695+00:00",
    "data_attrs": [
      {
        "template_id": "1",
        "created": "2018-01-25T12:30:42.167126+00:00",
        "label": "temperature",
        "value_type": "float",
        "type": "dynamic",
        "id": 1
      }
    ],
    "label": "Thermometer Template",
    "config_attrs": [],
    "attrs": [
      {
        "template_id": "1",
        "created": "2018-01-25T12:30:42.167126+00:00",
        "label": "temperature",
        "value_type": "float",
        "type": "dynamic",
        "id": 1
      },
      {
        "template_id": "1",
        "created": "2018-01-25T12:30:42.167126+00:00",
        "label": "fan",
        "type": "actuator",
        "value_type": "float",
        "id": 2
      }
    ],
    "id": 1
  }
}
```

Note que o template (modelo) ID é 1 (linha 35), caso você já tenha criado algum outro template este id será diferente.

Para criar um dispositivo baseado nesse modelo (ID 1), envie a seguinte requisição para a dojot

```
curl -X POST http://localhost:8000/device \
-H "Authorization: Bearer ${JWT}" \
-H 'Content-Type:application/json' \
-d ' {
  "templates": [
    "1"
  ],
  "label": "device"
}'
```

A lista de IDs de modelos na linha 6 contém um único ID do modelo configurado até o momento. Para conferir os dispositivos configurados, basta enviar uma requisição do tipo GET para /device:

```
curl -X GET http://localhost:8000/device -H "Authorization: Bearer ${JWT}"
```

Que deve retornar:

```
{
  "pagination": {
    "has_next": false,
    "next_page": null,
    "total": 1,
    "page": 1
  },
  "devices": [
    {
      "templates": [
        1
      ],
      "created": "2018-01-25T12:36:29.353958+00:00",
      "attrs": {
        "1": [
          {
            "template_id": "1",
            "created": "2018-01-25T12:30:42.167126+00:00",
            "label": "temperature",
            "value_type": "float",
            "type": "dynamic",
            "id": 1
          },
          {
            "template_id": "1",
            "created": "2018-01-25T12:30:42.167126+00:00",
            "label": "fan",
            "value_type": "actuator",
            "type": "float",
            "id": 2
          }
        ]
      },
      "id": "0998",
      "label": "device_0"
    }
  ]
}
```


10.4 Enviando mensagens

Até o momento um token de acesso foi obtido, um modelo e um dispositivo (baseado no modelo) foram criados. Em um sistema real, o dispositivo físico envia mensagens para a dojot com todos os seus atributos contendo valores correntes. Nsse tutorial serão enviadas mensagens MQTT montadas "na mão" para a plataforma, emulando um dispositivo físico. Para tal, será utilizado o mosquito_pub e mosquitto_sub do projeto mosquitto.

Atenção: Algumas distribuições Linux, distribuições Linux baseadas em Debian em particular, tem dois pacotes para mosquitto - um contendo ferramentas para cliente (ou seja, mosquitto_pub e mosquitto_sub para publicar mensagens e se subscrever tópicos) e outro contendo o broker MQTT também. E neste tutorial, apenas as ferramentas do pacote *mosquitto-clients* em Distribuições Linux baseadas no Debian serão usadas. Verifique se o broker MQTT não está em execução antes de iniciar o dojot (executando comandos como `ps aux | grep mosquitto`) para evitar conflitos de porta.

Nota: Para executar *mosquitto_pub* e *mosquitto_sub* sem usar TLS, como noexemplos abaixo, você deve definir ALLOW_UNSECURED_MODE com o valor‘ ‘true’‘ para o serviço *iotagent-mqtt*, isto é, *ALLOW_UNSECURED_MODE = ‘true’*.Você pode alterar este valor no arquivo *docker-compose.yml* da dojot e então matar e subir o docker-compose novamente. **Por padrão, esse valor já é ‘true’.**

O formato padrão de mensagem usado pela dojot é um simples “chave-valor” JSON (é possível traduzir qualquer formato para esse esquema utilizando fluxos), como abaixo:

```
{
  "temperature" : 10.6
}
```

Vamos enviar essa mensagem para a dojot:

```
mosquitto_pub -h localhost -t /admin/0998/attrs -p 1883 -i admin:0998 -m '{
↪"temperature": 10.6}'
```

Se não houver saída (output), a mensagem foi enviada ao broker MQTT.

Além disso, você pode enviar uma mensagem de configuração da dojot para o dispositivo. O atributo de destino deve ser do tipo “actuator” ou “atuador”.

Para simular o recebimento da mensagem em um dispositivo, podemos usar o mosquitto_sub:

```
mosquitto_sub -h localhost -p 1883  -i admin:0998 -t /admin/0998/config
```

Disparando o envio da mensagem da dojot para o dispositivo.

```
curl -X PUT \
    http://localhost:8000/device/0998/actuate \
    -H "Authorization: Bearer ${JWT}" \
    -H 'Content-Type:application/json' \
    -d '{"attrs": {"fan" : 100}}'
```

Como descrito no *Dúvidas Mais Frequentes*, existem algumas considerações a respeito dos tópicos MQTT:

- Pode-se configurar o ID do dispositivo origem da mensagem utilizando o parâmetro MQTT `client-id`. Deve seguir o seguinte padrão: `<service>:<deviceid>`, como em `admin:efac`.
- Se por algum motivo você não pode fazer tal coisa, então o dispositivo deve configurar seu ID no tópico utilizado para publicar as mensagens. O tópico deve assumir o padrão `/<service-id>/<device-id>/attrs` (por exemplo: `/admin/efac/attrs`).
- O tópico deve assumir o padrão `/<service-id>/<device-id>/config` (por exemplo: `/admin/efac/config`).
- Os dados da mensagem MQTT (payload) deve ser um JSON com cada chave sendo um atributo do dispositivo cadastrado na dojot, como:

```
{ "temperature" : 10.5,"pressure" : 770 }
```

Estes exemplos estão usando o MQTT sem TLS, nós recomendamos *Usando MQTT com segurança (TLS)*.

10.5 Conferindo dados históricos

A fim de se conferir todos os valores que foram enviados pelo dispositivo para um atributo particular, pode-se utilizar a API do History, veja mais em *Componentes e APIs*. Vamos, então, enviar agora alguns outros valores à dojot para que possamos conseguir resultados um pouco mais interessantes:

```
mosquitto_pub -t /admin/0998/attrs -i admin:0998 -m '{"temperature": 36.5}'
mosquitto_pub -t /admin/0998/attrs -i admin:0998 -m '{"temperature": 15.6}'
mosquitto_pub -t /admin/0998/attrs -i admin:0998 -m '{"temperature": 10.6}'
```

Para recuperar todos os valores enviados do atributo temperature desse dispositivo:

```
curl -X GET \
  -H "Authorization: Bearer ${JWT}" \
  "http://localhost:8000/history/device/0998/history?lastN=3&attr=temperature"
```

O endpoint do histórico é construído por meio desses valores:

- `.../device/0998/...`: o ID do dispositivo é `0998` - isso é obtido do atributo `id` do próprio dispositivo
- `.../history?lastN=3&attr=temperature`: o atributo requerido é temperature e deve ser recuperado os 3 últimos valores.

  A requisição deve resultar na seguinte mensagem:

```
[
  {
    "device_id": "0998",
    "ts": "2018-03-22T13:47:07.050000Z",
    "value": 10.6,
    "attr": "temperature"
  },
  {
    "device_id": "0998",
    "ts": "2018-03-22T13:46:42.455000Z",
    "value": 15.6,
    "attr": "temperature"
  },
  {
    "device_id": "0998",
    "ts": "2018-03-22T13:46:21.535000Z",
    "value": 36.5,
    "attr": "temperature"
  }
]
```

A mensagem acima contém todos os valores previamente enviados pelo dispositivo.

CAPÍTULO 11

Usando o construtor de fluxos (Flowbroker)

Este tutorial mostrará como usar corretamente o construtor de fluxo para processar mensagens e eventos gerados pelos dispositivos.

Nota:

- Para quem é: usuários iniciantes
- Nível: básico
- Tempo de leitura: 20 min

Table of Contents

11.1 Nós da dojot

- *Notificação (notification)*
- *Definir valor (Change)*
- *Interruptor (Switch)*
- *Modelo (Template)*
- *Cron*
- *Cron em lote (Cron batch)*
- *Geo referência (Geofence)*
- *Obter contexto (Get Context)*
- *Mesclar dados (Merge data)*
- *Soma acumulada (Cumulative sum)*
- *Nós descontinuados*
  - *Dispositivo (entrada) [Device in]*
  - *Modelo de dispositivo - entrada (Device template in)*
  - *Dispositivo (saída) [Device out]*
  - *Atuar (Actuate)*

11.1.1 Evento dispositivo - entrada (Device Event in)

Este nó especifica as mensagens recebidas de ou enviadas para um dispositivo específico. A mensagem criada por este nó é um pouco diferente daquela criada pelo nó DeviceIn:

```
{
    "data": {
        "attrs": {
            "temperature": 10,
            "some-static-attr": "efac"
        }
    }
}
```

Essa estrutura pode ser referenciada em nós como *Modelo do dispositivo*:

```
Sample message {{payload.data.attrs.temperature}}
```

Para configurar o dispositivo no nó, uma janela como a Fig. 11.1 será exibida.

Campos

- **Nome (Name)** *(opcional)*: Nome do nó
- **Dispositivo (Device)** *(obrigatório)*: o dispositivo *dojot* que acionará o fluxo
- **Eventos (Events)** *(obrigatório)*: selecione quais eventos acionarão esse fluxo. A opção *Atuação (Actuation)* seleciona mensagens de atuação (aquelas enviadas ao dispositivo) e *Publicação (Publication)* seleciona todas as mensagens publicadas pelo dispositivo.

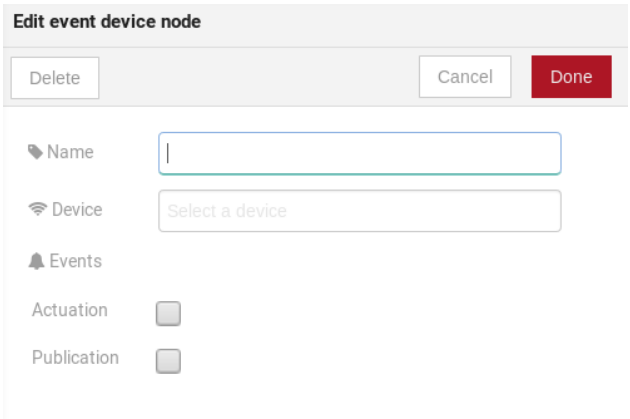
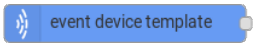


Fig. 11.1: : Dispositivo na janela de configuração

Nota: Se o dispositivo que aciona um fluxo for removido, o fluxo não funcionará mais.

11.1.2 Modelo de dispositivo - Eventos - entrada (event device template)



Este nó especifica que as mensagens dos dispositivos compostos por um modelo específico acionarão esse fluxo. Por exemplo, se o modelo de dispositivo definido neste nó for o modelo A, todos os dispositivos compostos com o modelo A acionarão o fluxo. Por exemplo: dispositivo1 é composto pelos modelos [A, B], dispositivo2 pelo modelo A e dispositivo3 pelo modelo B. Então, nesse cenário, apenas as mensagens do dispositivo1 e do dispositivo2 iniciarão o fluxo, porque o modelo A é um dos modelos que compor esses dispositivos.

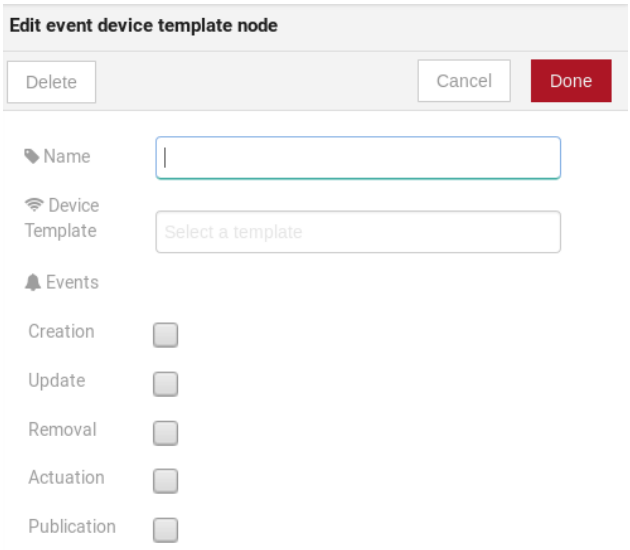


Fig. 11.2: : Modelo de dispositivo janela de configuração

Campos

- **Nome (Name)** *(opcional)*: Nome do nó.
- **Dispositivo (Device)** *(obrigatório)*: o dispositivo *dojot* que acionará o fluxo.
- **Eventos (Events)** *(obrigatório)*: Selecione qual evento acionará esse fluxo. A *criação (Creation)*, a *atualização (Update)* e a *remoção (Removal)* estão relacionadas às operações de gerenciamento de dispositivos. *Atuação (Actuation)* acionará esse fluxo no caso de enviar mensagens de atuação para o dispositivo e *Publicação (Publication)* acionará esse fluxo sempre que um dispositivo publicar uma mensagem para executar.

11.1.3 Saída para multiplos dispositivos (Multi device out)

A saída do dispositivo determinará qual dispositivo (ou dispositivos) terá seus atributos atualizados no dojot de acordo com o resultado do fluxo. Lembre-se de que este nó não envia mensagens para o seu dispositivo; ele atualiza apenas os atributos na plataforma. Normalmente, o dispositivo escolhido é um dispositivo virtual, que existe apenas no dojot.

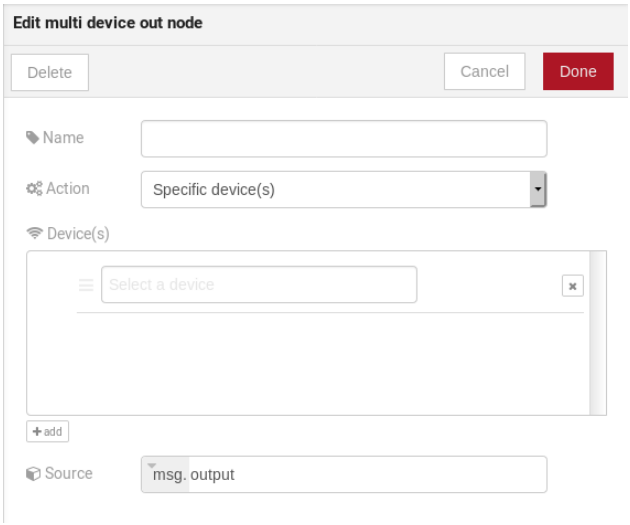


Fig. 11.3: : Janela de configuração do dispositivo

Campos

- **Nome (Name)** *(opcional)*: Nome do nó.
- **Ação (Action)** ***(obrigatório)*: Qual nó receberá a atualização. As opções são:**
  - O dispositivo que acionou o fluxo: isso atualizará o mesmo dispositivo que enviou a mensagem que acionou esse fluxo.
  - *Dispositivo(s) específicos (Specific device(s))*: quais nós que receberão a atualização.
  - *Dispositivo(s) definido(s) durante o fluxo (Device(s) defined during the flow)*: quais nós que receberão a atualização. Isso é referenciado por uma lista de valores, assim como os valores de saída (msg.list_of_devices).
- **Dispositivo (Device)** *(obrigatório)*: selecione "O dispositivo que acionou o fluxo" fará com que o dispositivo que foi o ponto de entrada seja o ponto final do fluxo. "Dispositivo específico" qualquer dispositivo escolhido será a saída do fluxo e "um dispositivo definido durante o fluxo" tornará um dispositivo que o fluxo selecionou durante a execução o ponto final.
- **Origem (Source)** *(obrigatório)*: estrutura de dados que será mapeada como mensagem para saída do dispositivo

11.1.4 Multi atuador (Multi actuate)

O nó de atuação é, basicamente, a mesma coisa que o nó de dispositivo (saída). Mas pode enviar mensagens para um dispositivo real, como dizer a uma lâmpada para desligar a luz e etc.

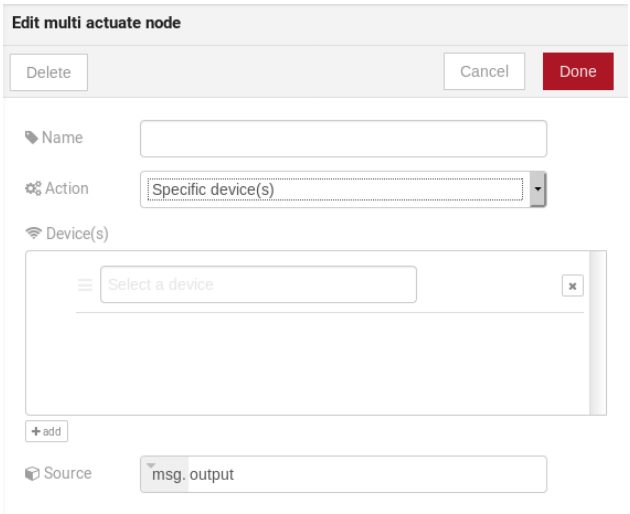
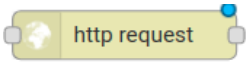


Fig. 11.4: : Configuração de atuação

Campos

- **Nome (Name)** *(opcional)*: Nome do nó.
- **Ação (Action)** ***(obrigatório)*****: para qual dispositivo uma mensagem será enviada. As opções são:**
  - O dispositivo que acionou o fluxo: isso enviará uma mensagem para o mesmo dispositivo que enviou a mensagem que acionou esse fluxo.
  - *Dispositivo(s) específicos (Specific device(s))*: para qual nó a mensagem será enviada.
  - *Dispositivos definidos durante o fluxo (Device(s) defined during the flow)*: para quais nós a mensagem será enviada. Isso é referenciado por uma lista de valores, assim como os valores de saída (msg.list_of_devices).
- **Dispositivo (Device)** *(obrigatório)*: selecione "O dispositivo que acionou o fluxo" fará com que o dispositivo que foi o ponto de entrada seja o ponto final do fluxo. "Dispositivo específico" qualquer dispositivo escolhido será a saída do fluxo e "um dispositivo definido durante o fluxo" tornará um dispositivo que o fluxo selecionou durante a execução o ponto final.
- **Origem (Source)** *(obrigatório)*: estrutura de dados que será mapeada como mensagem para saída do dispositivo

11.1.5 HTTP request



Este nó envia uma solicitação http para um determinado endereço e, em seguida, pode encaminhar a resposta para o próximo nó no fluxo.

Campos

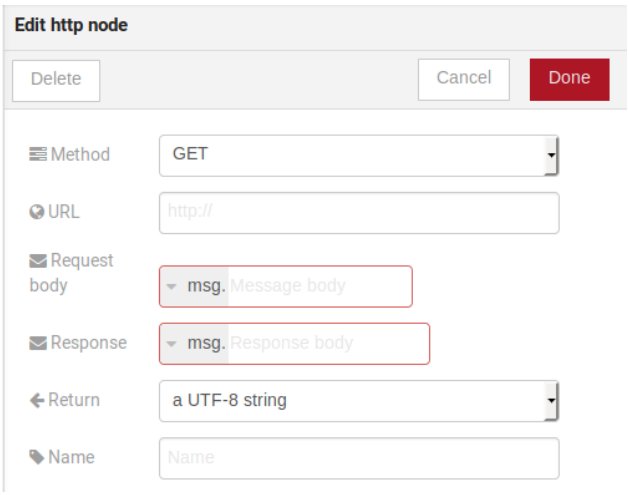


Fig. 11.5: : Configuração do Http Request

- **Método (Method)** *(obrigatório)*: o método http (GET, POST, etc . . .).
- **URL** *(obrigatório)*: a URL que receberá a solicitação http
- **Corpo da solicitação (Request body)** *(obrigatório)*: variável que contém o corpo da solicitação. Este valor pode ser atribuído à variável usando o nó do modelo, por exemplo.
- **Resposta (Response)** *(obrigatório)*: variável que receberá a resposta http.
- **Retorno (Return)** *(obrigatório)*: Tipo do retorno.
- **Nome (Name)** *(opcional)*: Nome do nó.

11.1.6 Ftp request

Este nó envia um arquivo para um servidor FTP. Ao fazer upload de um arquivo, seu nome pode ser definido configurando o campo "Nome do arquivo" da mesma maneira que outras variáveis de saída (ele deve se referir a uma variável definida no fluxo). A codificação do arquivo definirá o *enconding* do arquivo, que pode ser, por exemplo, "base64" ou "utf-8".

Campos

- **Método (Method)** *(obrigatório)*: a ação do FTP a ser executada (PUT).
- **URL** *(obrigatório)*: o servidor FTP
- **Autenticação (Authentication)** *(obrigatório)*: Nome de usuário e senha para acessar este servidor.
- **Nome do arquivo (File name)** *(obrigatório)*: variável que contém o nome do arquivo a ser carregado.
- **Conteúdo do arquivo (File content)** *(obrigatório)*: Essa variável deve conter o conteúdo do arquivo.
- **Codificação de arquivo (File encoding)** *(obrigatório)*: como o arquivo é codificado
- **Resposta (Response)** *(obrigatório)*: variável que receberá a resposta http.
- **Nome (Name)** *(opcional)*: Nome do nó.

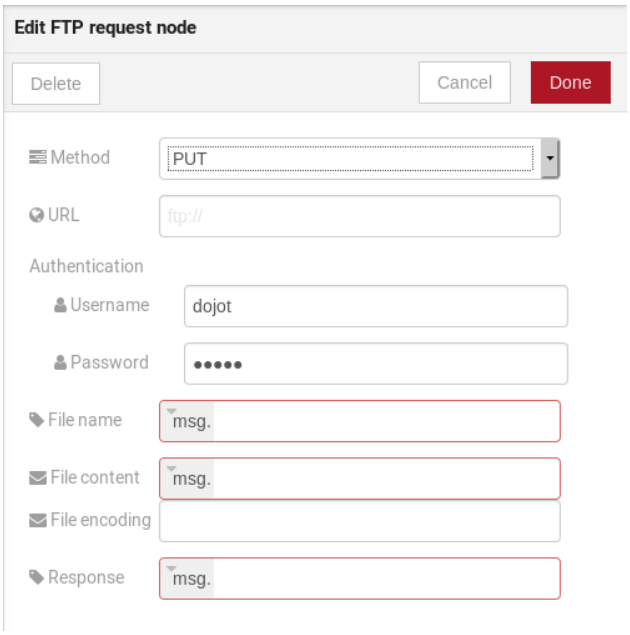


Fig. 11.6: : Modelo de dispositivo janela de configuração

11.1.7 Notificação (notification)

Este nó envia uma notificação do usuário para outros serviços na dojot. Isso pode ser útil para gerar notificações de aplicativos que podem ser consumidas por uma aplicação front-end. O usuário pode definir uma mensagem estática a ser enviada ou, como outros nós de saída, configurar um conjunto de variáveis em um nó anterior que será resolvido no tempo de execução. Além disso, os metadados podem ser adicionados à mensagem: pode ser um objeto de valor-chave simples que contém dados arbitrários.

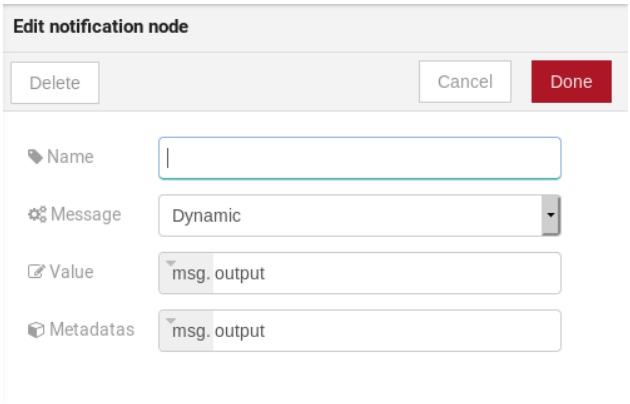


Fig. 11.7: : Modelo de dispositivo janela de configuração

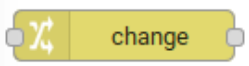
Campos

- **Nome (Name)** *(opcional)*: Nome do nó
- **Mensagem (Message)** *(obrigatório)*: Estática, se a notificação contiver um texto estático ou dinâmico, que permitirá que uma variável seja configurada como saída para este nó. Essa variável será substituída no tempo

de execução.

- **Valor (Value)** *(obrigatório)*: conteúdo da mensagem (texto estático ou referência variável).
- **Metadados (Metadata)** *(obrigatório)*: referência de variável que contém um dicionário simples (pares de valores-chave) que contém os metadados a serem adicionados à mensagem

11.1.8 Definir valor (Change)



O nó de Definir valor é usado para copiar ou atribuir valores a uma saída, por exemplo, copie valores de atributos de uma mensagem para um dicionário que será atribuído ao dispositivo virtual.

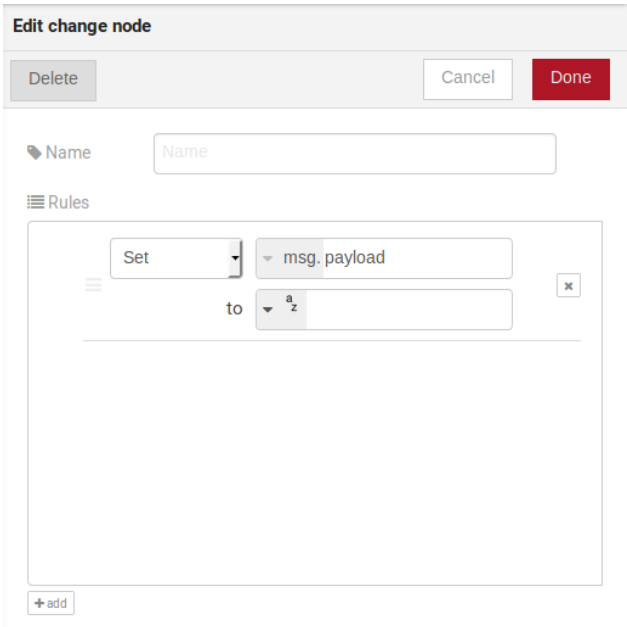


Fig. 11.8: : Definir valor (Change) configuração

Campos

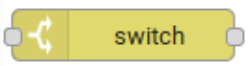
- **Nome (Name)** *(opcional)*: Nome do nó
- **msg** *(obrigatório)*: definição da estrutura de dados que será enviada para o próximo nó e receberá o valor definido no campo para
- **para (to)** *(obrigatório)*: atribuição ou cópia de valores

---

Nota: Mais de uma regra pode ser atribuída clicando em *+adicionar* abaixo da caixa de regras

---

11.1.9 Interruptor (Switch)



O nó Switch permite que as mensagens sejam roteadas para diferentes ramificações de um fluxo avaliando um conjunto de regras em relação a cada mensagem.

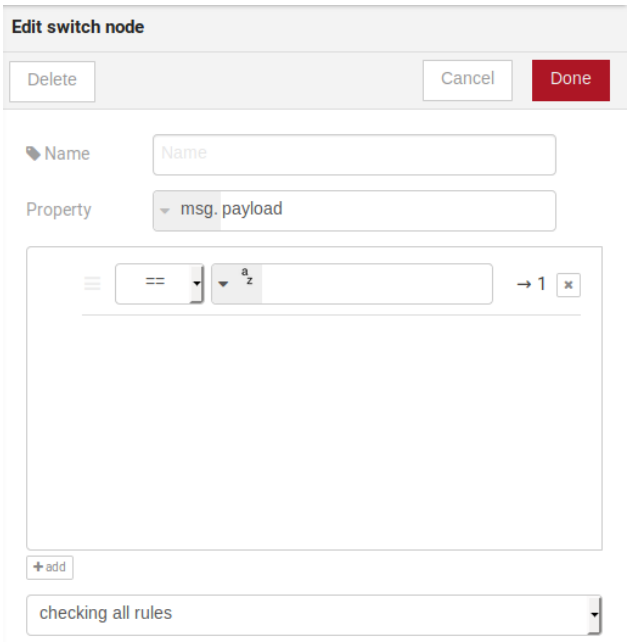


Fig. 11.9: : Switch configurações

Campos

- **Nome (Name)** *(opcional)*: Nome do nó
- **Propriedade (Property)** *(obrigatório)*: variável que será avaliada
- **Caixa de regras** *(obrigatório)*: regras que determinarão a ramificação de saída do nó. Além disso, ele pode ser configurado para interromper a verificação de regras quando encontrar uma que corresponda a outra ou verificar todas as regras e rotear a mensagem para a saída correspondente.

Nota:

- Mais de uma regra pode ser atribuída clicando em *+adicionar* abaixo da caixa de regras
- As regras são mapeadas individualmente para os conectores de saída. Que a primeira regra está relacionada à primeira saída, a segunda regra à segunda saída e etc...

11.1.10 Modelo (Template)

Nota: Apesar do nome, este nó não tem nada a ver com modelos de dispostivos da dojot

---

Este nó atribuirá um valor a uma variável de destino. Este valor pode ser uma constante, o valor de um atributo que veio do dispositivo de entrada e etc.

Ele usa a linguagem mustache. Verifique a Fig. 11.10 como exemplo: o campo **a** da *payload* será substituído pelo valor de **payload.b**

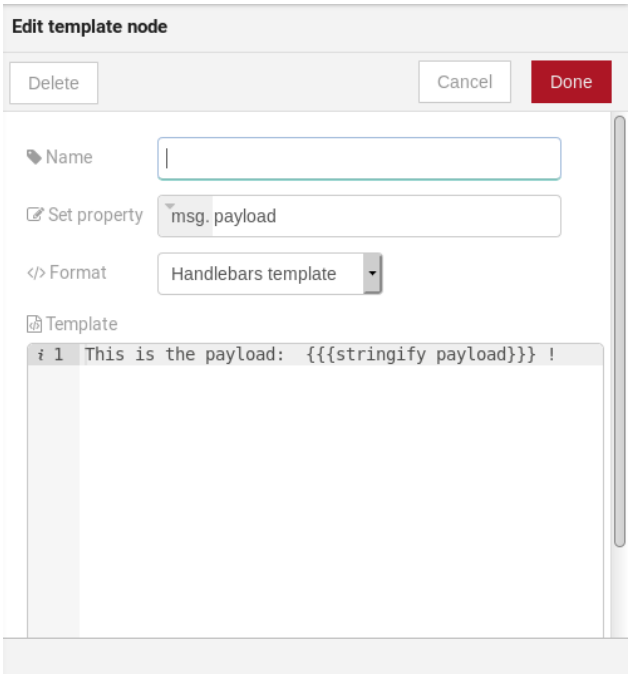
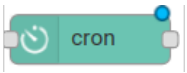


Fig. 11.10: : Configurações do Modelo

Campos

- **Nome (Name)** *(opcional)*: Nome do nó
- **Propriedade (Set Property)** *(obrigatório)*: variável que receberá o valor
- **Formato (Format)** *(obrigatório)*: o modelo de formato que será gravado
- **Modelo (Template)** *(obrigatório *: valor que será atribuído à variável de destino definida em **Propriedade**
- **Saída (Output as)** *(obrigatório)*: o formato da saída

11.1.11 Cron



Nó de processamento para criar/remover tarefas cron. Cron permiti agendar tarefas para: enviar eventos para o data broker ou executar uma requisição http.

Campos

- **Operação** *(obrigatório)*: define o tipo de processamento ao criar ou remover trabalhos cron (CREATE, REMOVE).
- **Expressão de Tempo do CRON** *(obrigatório)*: Expressão de Tempo do CRON, por exemplo * * * * * *. Obrigatório ao usar a operação do tipo CREATE.
- **Nome do Trabalho** *(opcional)*: Nome do JOB.
- **Descrição do Trabalho** *(opcional)*: Descrição do Trabalho.

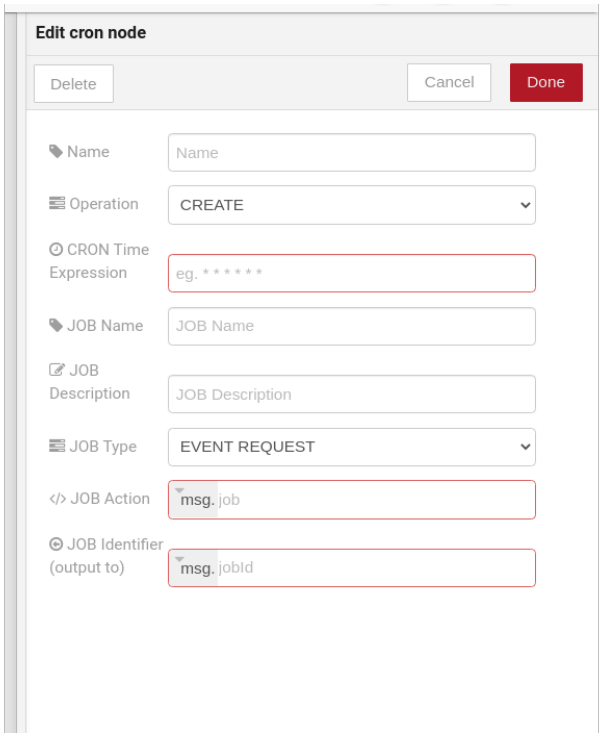


Fig. 11.11: : Cron configuração

- **Tipo do Trabalho** *(obrigatório)*: As opções são EVENT REQUEST ou HTTP REQUEST.
- **Ação do Trabalho** *(obrigatório)*: Variável que contém o JSON para Ação do Trabalho. Este valor pode ser atribuído à variável usando o nó do modelo, por exemplo.
- **Identificador do Trabalho (saída para)** *(obrigatório)*: Variável que receberá o Identificador do Trabalho.
- **Nome (Name)** *(opcional)*: Nome do nó

Exemplo de *Ação do Trabalho* quando *Tipo do Trabalho* é **HTTP REQUEST**:

```
{
    "method": "PUT",
    "headers": {
                  "Authorization": "Bearer ${JWT}",
                  "Content-Type": "application/json"
                },
    "url": "http://device-manager:5000/device/${DEVICE_ID}/actuate",
    "body": {
                "attrs": {"message": "keepalive"}
            }
}
```

Exemplo de *Ação do Trabalho* quando *Tipo do Trabalho* é **EVENT REQUEST**:

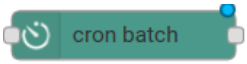
```
{
    "subject": "dojot.device-manager.device",
    "message": {
                  "event": "configure",
                  "data": { "attrs": { "message": "keepalive"},
```

(continues on next page)

(continuação da página anterior)

```
                                "id": "6a1213"
                              },
                    "meta": { "service": "admin"}
                  }
}
```

11.1.12 Cron em lote (Cron batch)



Funciona como o *cron node*, mas aqui você pode usar agendamentos em lote.

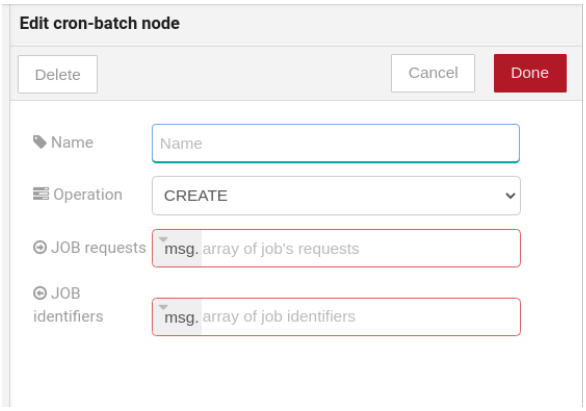


Fig. 11.12: : Cron batch configurações

Campos

- **Operação** *(obrigatório)*: define o tipo de processamento ao criar ou remover trabalhos cron (CREATE, REMOVE).
- **Requisições de trabalho** *(obrigatório)*: Variável que contém o *array* de JSONs para ações de trabalho
- **Identificadores de trabalho** *(obrigatório)*: Variável que receberá o *array* de identificadores de trabalho.
- **Nome (Name)** *(opcional)*: Nome do nó

11.1.13 Geo referência (Geofence)

Select an interest area to determine wich devices will activate the flow

Campos

- **Área (Area)** *(obrigatório)*: Área que será selecionada. Pode ser escolhido com um quadrado ou um pentago
- **Filtro (Filter)** *(obrigatório)*: Qual lado da área será selecionado: dentro ou fora da área marcada no campo acima.
- **Nome (Name)** *(opcional)*: Nome do nó

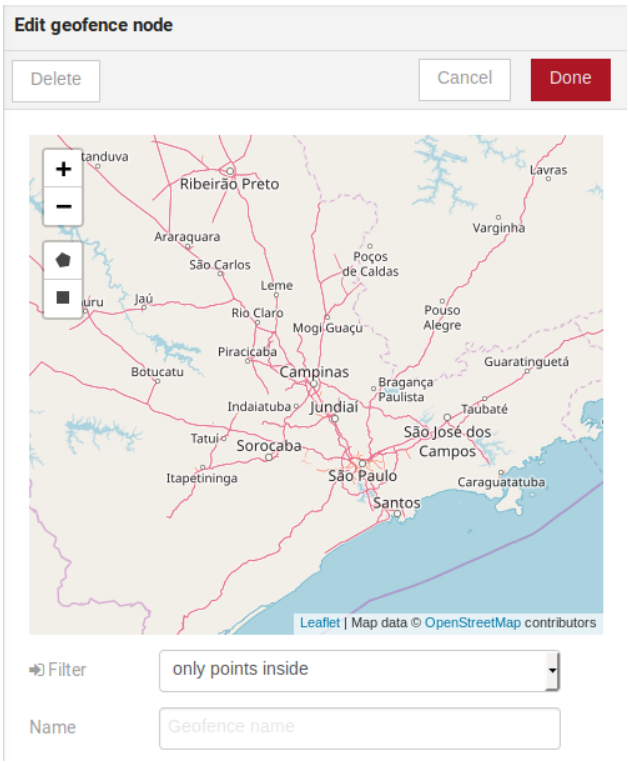
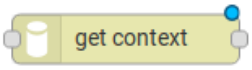


Fig. 11.13: : Geo referência (Geofence) configuração

11.1.14 Obter contexto (Get Context)



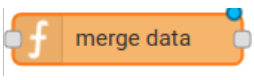
Este nó é usado para obter uma variável que está no contexto e atribuir seu valor a uma variável que será usada no fluxo.

Nota: O contexto é um mecanismo que permite que um determinado conjunto de dados persista além da vida do evento, possibilitando o armazenamento de um estado para os elementos da solução.

Campos

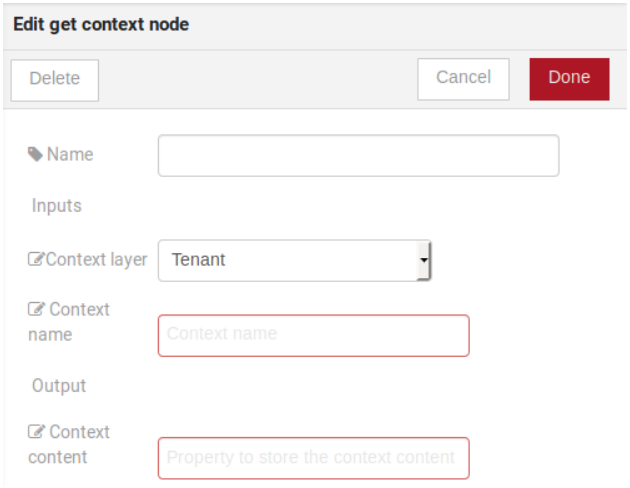
- **Nome (Name)** *(opcional)*: Nome do nó
- **Camada de contexto (Context layer)** *(obrigatório)*: a camada do contexto em que a variável está
- **Nome do contexto (Context name)** *(obrigatório)*: a variável que está no contexto
- **Conteúdo do contexto (Context content)** *(obrigatório)*: a variável no fluxo que receberá o valor do contexto

11.1.15 Mesclar dados (Merge data)



Este nó permite que os objetos sejam mesclados no contexto.

Campos



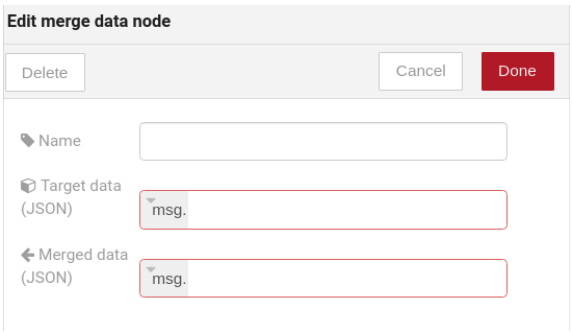
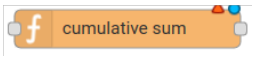


Fig. 11.14: : Configuração de Mesclar dados

- **Dado alvo (JSON)** *(required)*: Variável que contém os dados a serem mesclados.
- **Dado mesclado (JSON) (JSON)** *(required)*: Variável que receberá os novos dados mesclados com os dados existentes.
- **Nome (Name)** *(opcional)*: Nome do nó

11.1.16 Soma acumulada (Cumulative sum)



O nó de soma cumulativa acumula os dados para um atributo em uma janela temporal e mantém isso no contexto.

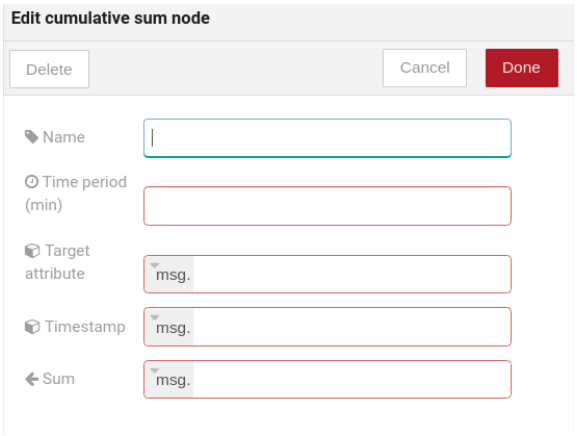


Fig. 11.15: : Configuração de Soma acumulada

Campos

- **Período de tempo (min)** *(obrigatório)*: Tempo em minutos para manter a soma.
- **Atributo alvo** *(obrigatório)*: Variável que contém o valor a ser soma.
- **Tempo do evento** *(Obrigatório)*: Variável que contém o timestamp advindo do device ou da dojot. Na maioria das vezes pode ser setado com *payload.metadata.timestamp*.
- **Somatório** *(Obrigatório)*: Variável que receberá a soma.
- **Nome (Name)** *(opcional)*: Nome do nó

11.1.17 Nós descontinuados

Esses nós estão programados para serem removidos em versões futuras. Eles funcionarão sem problemas com os fluxos atuais.

Dispositivo (entrada) [Device in]

Este nó determina um dispositivo específico para ser o ponto de entrada de um fluxo. Para configurar o dispositivo no nó, uma janela como a Fig. 11.16 será exibida.

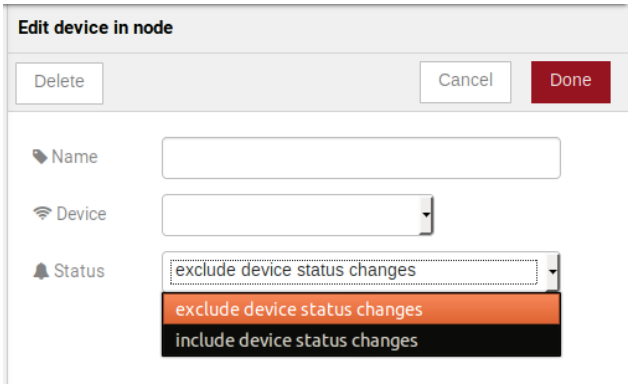


Fig. 11.16: : Dispositivo na janela de configuração

Campos

- **Nome (Name)** *(opcional)*: Nome do nó
- **Dispositivo (Device)** *(obrigatório)*: o dispositivo *dojot* que acionará o fluxo
- **Estado (Status0** *(obrigatório)*: excluir alterações de status do dispositivo não usará alterações de status do dispositivo (online, offline) para acionar o fluxo. Por outro lado, incluir alterações de status dos dispositivos usará esses status para acionar o fluxo.

Nota: Se o dispositivo que aciona um fluxo for removido, o fluxo se tornará inválido.

Modelo de dispositivo - entrada (Device template in)

Esse nó fará com que um fluxo seja acionado por dispositivos compostos por um determinado modelo. Se o modelo de dispositivo configurado no **modelo de dispositivo** no nó for o modelo A, todos os dispositivos compostos com o modelo A acionarão o fluxo. Por exemplo: *dispositivo1* é composto pelos modelos [A, B], *dispositivo2* pelo modelo A e *dispositivo3* pelo modelo B. Então, nesse cenário, apenas as mensagens do *dispositivo1* e do *dispositivo2* iniciarão o fluxo, porque o modelo A é um dos modelos que compor esses dispositivos.

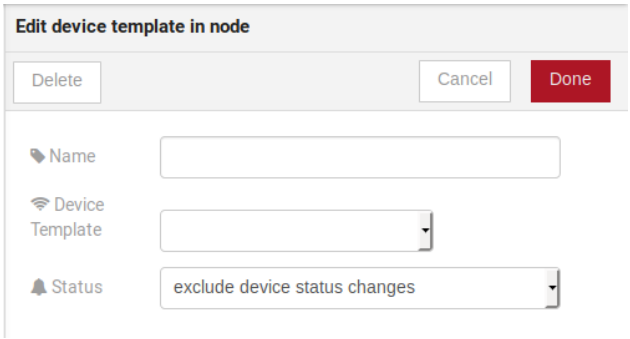
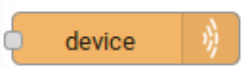


Fig. 11.17: : Modelo de dispositivo janela de configuração

Campos

- **Nome (Name)** *(opcional)*: Nome do nó.
- **Dispositivo (Device)** *(obrigatório)*: o dispositivo *dojot* que acionará o fluxo.
- **Estado (Status)** *(obrigatório)*: escolha se as alterações de status dos dispositivos acionarão ou não o fluxo.

Dispositivo (saída) [Device out]



A saída do dispositivo determinará qual dispositivo terá seus atributos atualizados no dojot de acordo com o resultado do fluxo. Lembre-se de que este nó não envia mensagens para o seu dispositivo; ele atualiza apenas os atributos na plataforma. Normalmente, o dispositivo escolhido é um dispositivo virtual, que existe apenas na dojot.

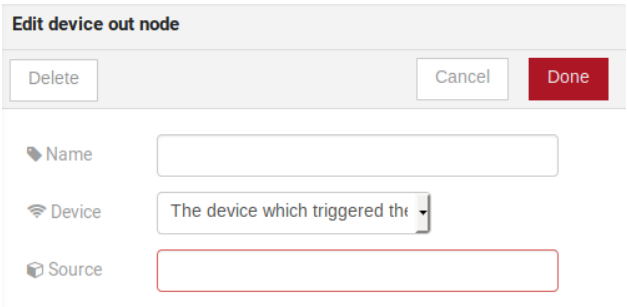


Fig. 11.18: : Janela de configuração do dispositivo

Campos

- **Nome (Name)** *(opcional)*: Nome do nó.
- **Dispositivo (Device)** *(obrigatório)*: selecione "O dispositivo que acionou o fluxo" fará com que o dispositivo que foi o ponto de entrada seja o ponto final do fluxo. "Dispositivo específico" qualquer dispositivo escolhido será a saída do fluxo e "um dispositivo definido durante o fluxo" tornará um dispositivo que o fluxo selecionou durante a execução o ponto final.
- **Origem (Source)** *(obrigatório)*: estrutura de dados que será mapeada como mensagem para saída do dispositivo

Atuar (Actuate)

O nó de atuação é, basicamente, a mesma coisa que o nó de dispositivo (saída). Mas pode enviar mensagens para um dispositivo real, como dizer a uma lâmpada para desligar a luz e etc.

Campos

- **Nome (Name)** *(opcional)*: Nome do nó.
- **Dispositivo (Device)** *(obrigatório)*: um dispositivo real no dojot
- **Origem (Source)** *(obrigatório)*: estrutura de dados que será mapeada como mensagem para saída do dispositivo

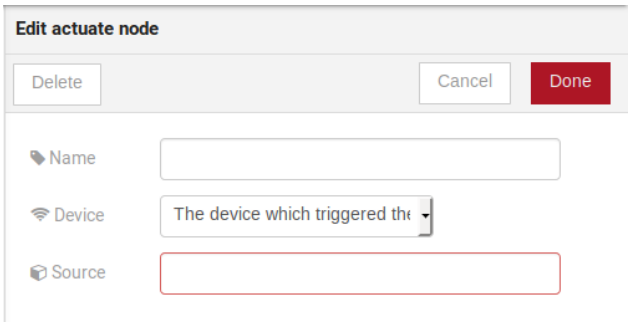


Fig. 11.19: : Configuração de atuação

11.2 Aprenda por exemplos

- *Usando nó http*
- *Usando o nó de geo referência (geofence)*
- *Usando os nós de soma acumulada, interruptor and notificação*

11.2.1 Usando nó http

Imagine este cenário: um dispositivo envia um *usuário* e uma *senha* e, a partir desses atributos, o fluxo solicitará a um servidor um token de autenticação que será enviado a um dispositivo virtual que possua um atributo de *token*.

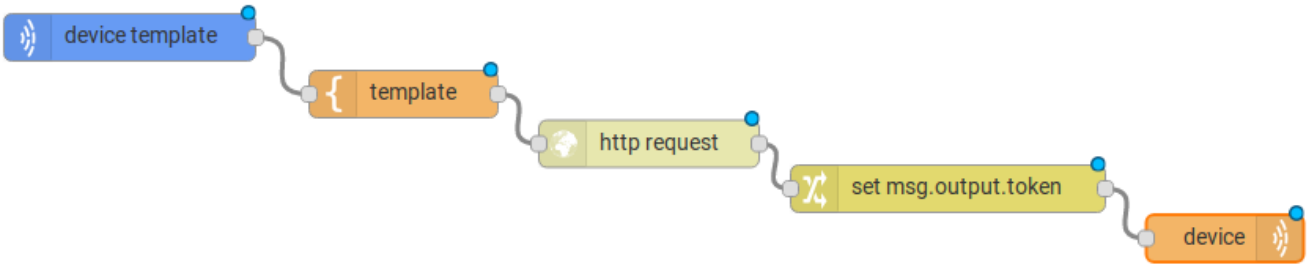


Fig. 11.20: : Fluxo usado para explicar o nó http

Para enviar essa solicitação ao servidor, o método http deve ser um POST e os parâmetros devem estar dentro da requisição. Portanto, no nó do modelo, um objeto JSON será designado a uma variável. O corpo (parâmetros *usuário* e *senha*) da requisição será designado à chave de *payload* do objeto JSON. E, se necessário, esse objeto também pode ter um *headers* como chave.

Em seguida, no nó http, o campo Requisição receberá o valor do objeto criado no nó do modelo. E, a resposta será

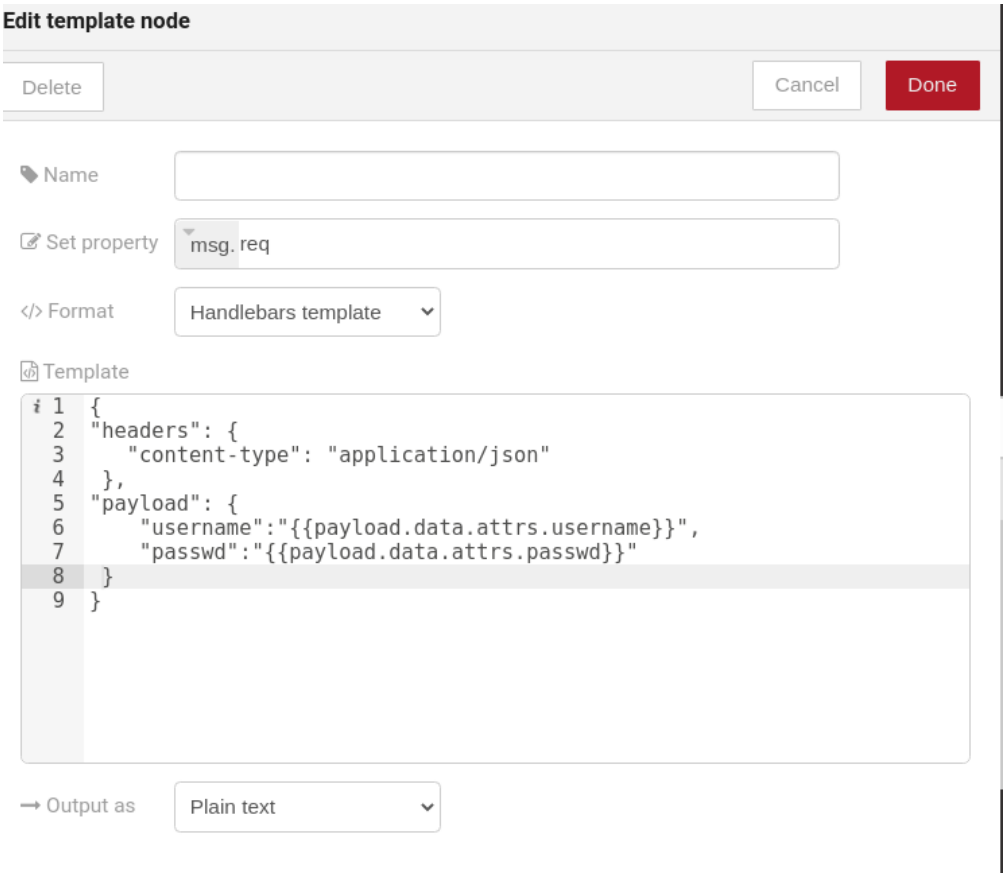


Fig. 11.21: : Configuração do nó do modelo

atribuída a qualquer variável, neste caso, é *msg.res*.

Nota: Se o buffer UTF-8 String for escolhido no campo de retorno, o corpo do corpo da resposta será uma string. Se o objeto JSON for escolhido, o corpo será um objeto.

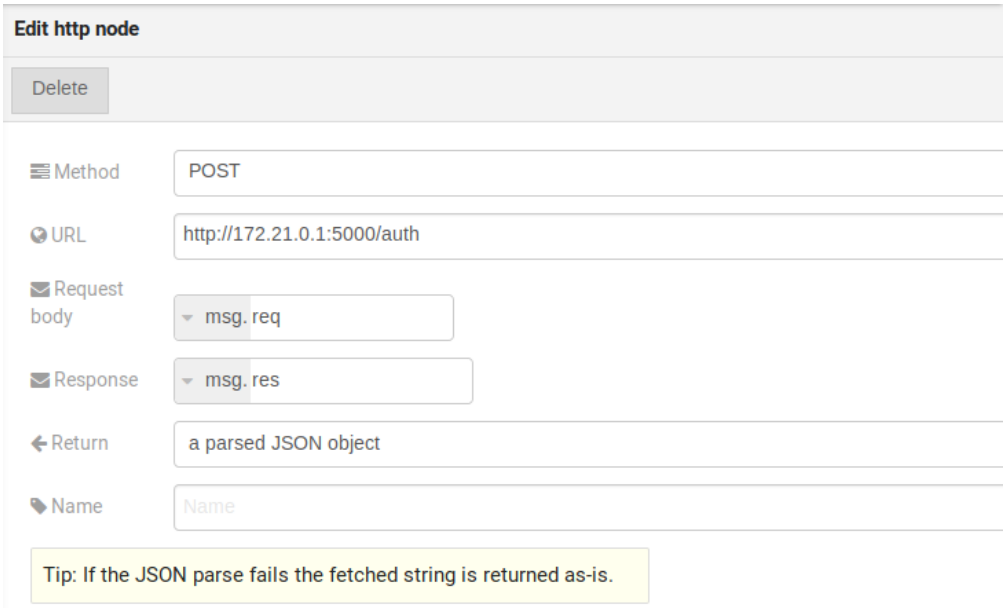


Fig. 11.22: : Configuração do nó do modelo

Como visto, a resposta do servidor é *req.res* e o corpo da resposta pode ser acessado em **msg.res.payload**. Portanto, as chaves do objeto que veio na resposta podem ser acessadas por: **msg.res.payload.key**. Na figura Fig. 11.23, o token que veio na resposta é atribuído ao token de atributo do dispositivo virtual.

Em seguida, o resultado do fluxo é que o *token* de atributo do dispositivo virtual seja atualizado com o token que veio na resposta da solicitação http:

11.2.2 Usando o nó de geo referência (geofence)

Um bom exemplo para aprender como o nó geo referência (geofence) funciona é com o fluxo abaixo:

O nó de geo referência é definido como parece na Fig. 11.27. A única coisa que diferencia os nós de geo referência *in area* e *out of the area* é o campo *Filtro* que, no primeiro, é configurado para apontar apenas para dentro e para fora no segundo, respectivamente.

Em seguida, se o dispositivo definido como dispositivo enviar uma mensagem com um atributo geográfico, o nó geo referência avaliará o ponto geográfico de acordo com sua regra e se corresponder à regra, o nó encaminhará as informações para o próximo nó e, se não , a execução da ramificação, que possui a geo referência que a regra não corresponde, para.

Nota: Para o nó de geo referência funcionar, a mensagem recebida deve ter um atributo geográfico; caso contrário, as ramificações do fluxo serão interrompidas nos nós de geo referência.

De volta ao exemplo, se o carro enviar uma mensagem de que ele está na área marcada, como `{"position": "-22.820156, -47.2682535"}`, a mensagem recebida no dispositivo será “Carro está dentro da área marcada”

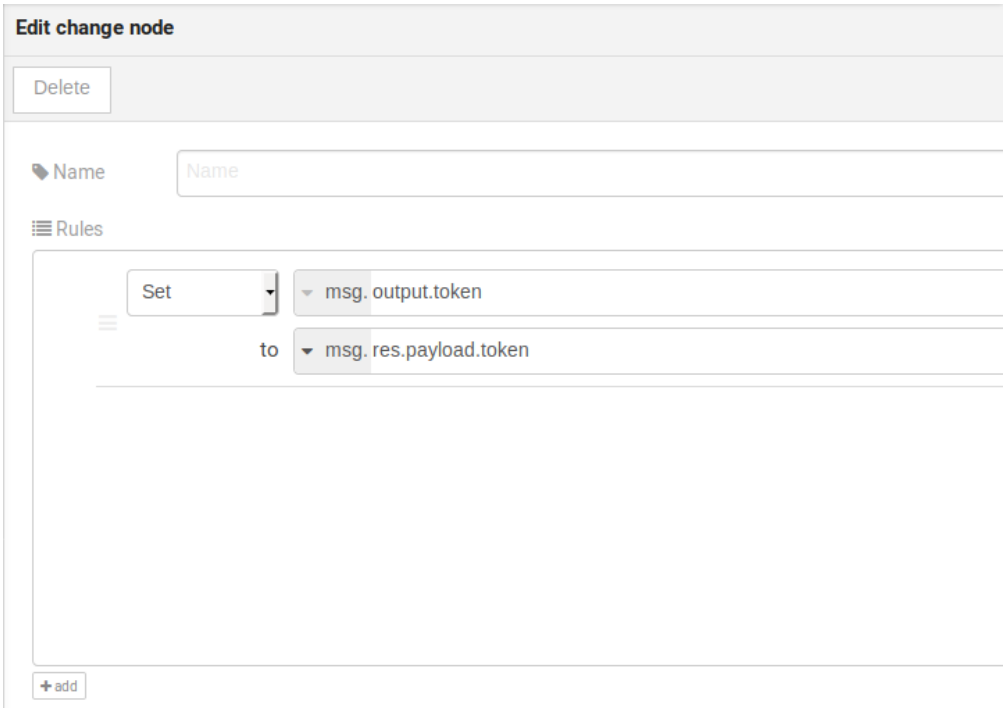


Fig. 11.23: : Configuração do nó do modelo

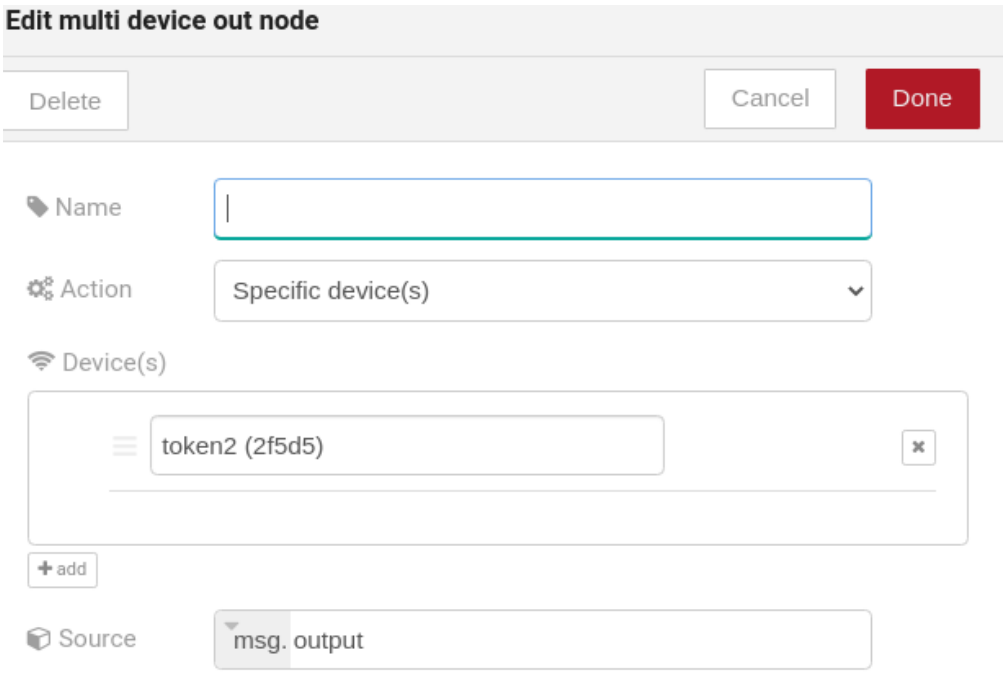


Fig. 11.24: : Configuração do dispositivo de saída

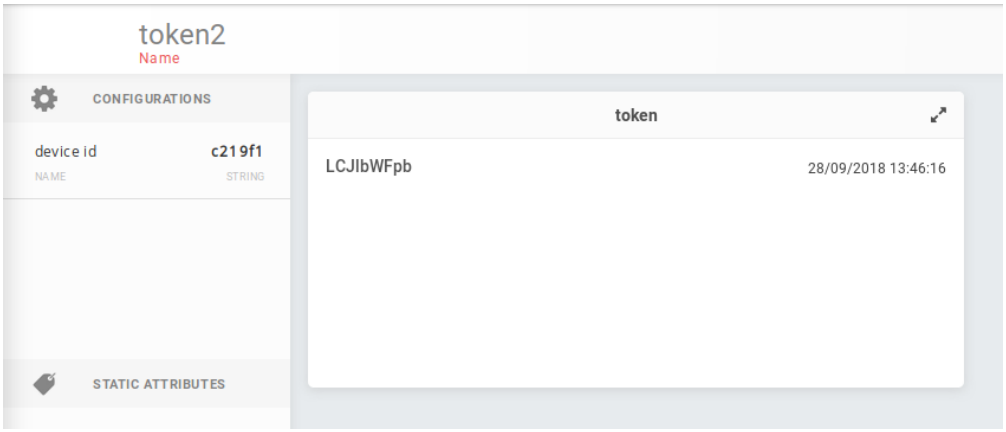


Fig. 11.25: : Dispositivo atualizado

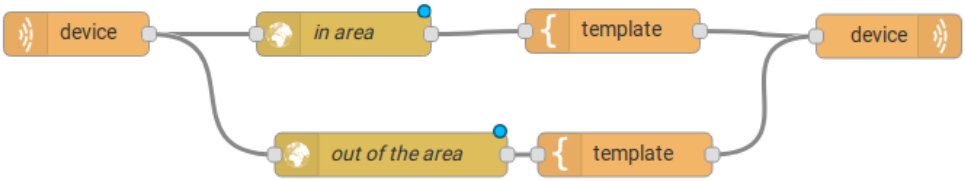


Fig. 11.26: : Fluxo usando geo referência

e se enviar `{"position": "0,0"}` o dispositivo receberá "O carro está fora da área marcada"

11.2.3 Usando os nós de soma acumulada, interruptor and notificação

Imagine este cenário: um dispositivo envia o nível de chuva, queremosgerar uma notificação se o acumulado, soma, das chuvas nas últimashora é maior que 100.

No nó *cumulative sum*, nós iremos acumular o valor de *rain* (*Target attribute*) na janela de tempo de 60 minutos (*Time period*) e iremos setar essa soma em um novo atributo chamado *payload.data.attrs.rain60Min* (*Sum*). A configuração *Timestamp* refere-se ao timestamp advindo do device ou da dojot. Na maioria das vezes pode ser setado com *payload.metadata.timestamp*. Veja mais em Fig. 11.31

Nós queremos que a notificação seja disparada apenas se o valor de chuva acumulado seja maior que 100, para isso usaremos o nó switch. Como na imagem Fig. 11.32.

Agora, caso nosso valor seja maior que 100 precisamos gerar a notificação, para tal usaremos um nó auxiliar antes, o nó *template*. No nó template iremos criar a mensagem que irá aparecer na notificação e definir seus metados, Fig. 11.33

Finalmente, vamos configurar o nó de notificação, como na imagem Fig. 11.34.

Portanto, se a estação meteorológica (dispositivo definido no nó do dispositivo de evento compublicação marcada) envia várias mensagens como *{"rain": 5}* durantea última hora e em uma dessas vezes a soma for superior a 100, ouma notificação será gerada. Nota: Várias notificações podem sergerado, desde que o valor acumulado seja superior a 100 na última hora. Vejaimage Fig. 11.35.

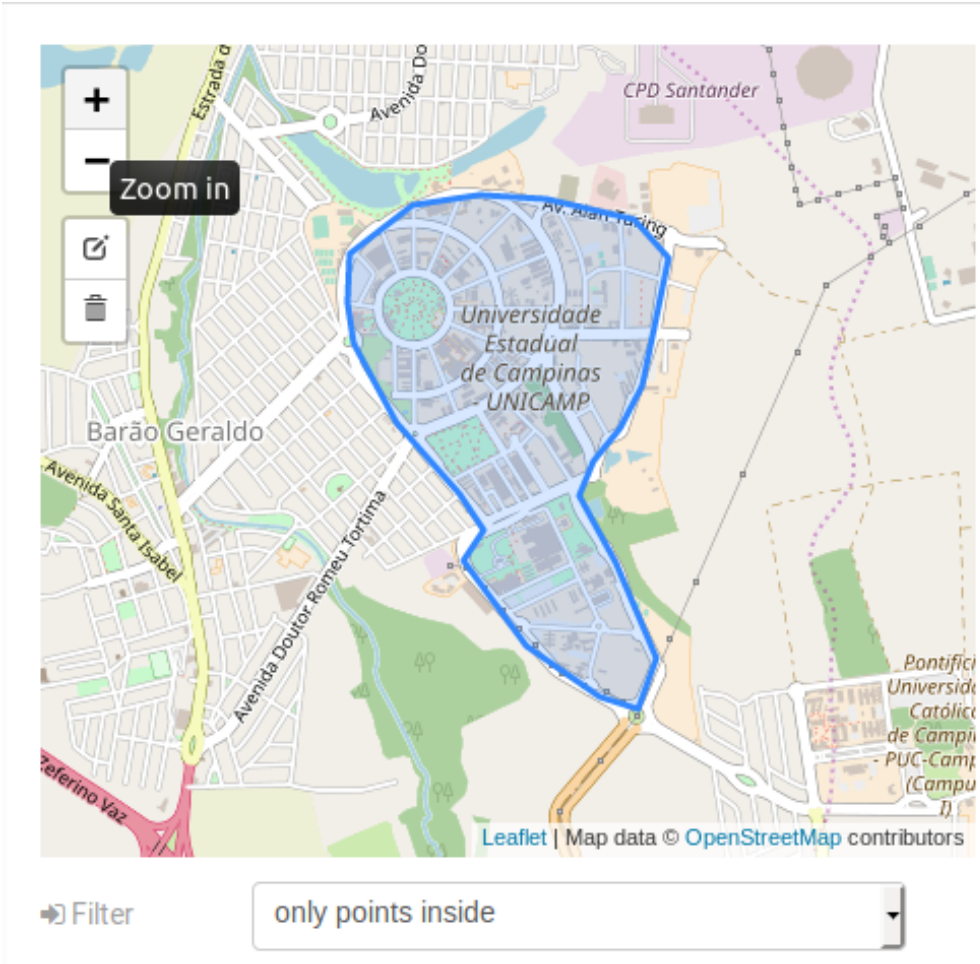


Fig. 11.27: : Configuração do nó de geo referência

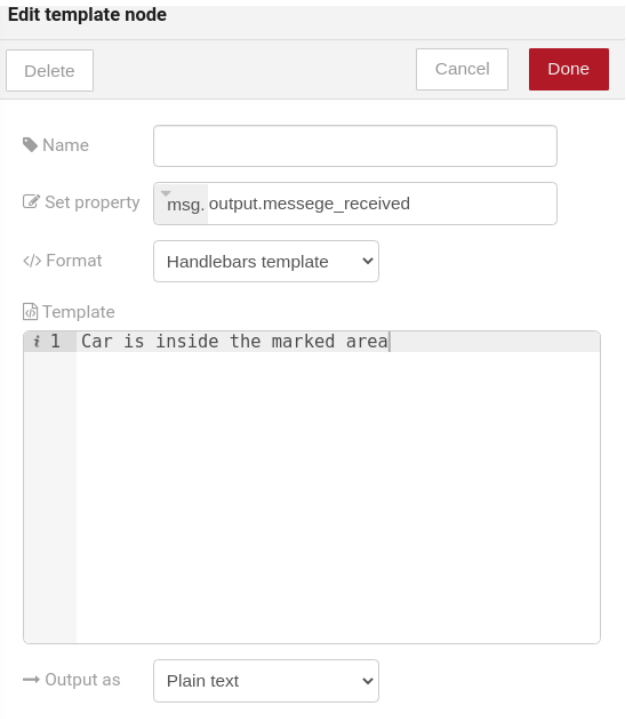


Fig. 11.28: : Configuração do nó do modelo se o carro estiver na área marcada

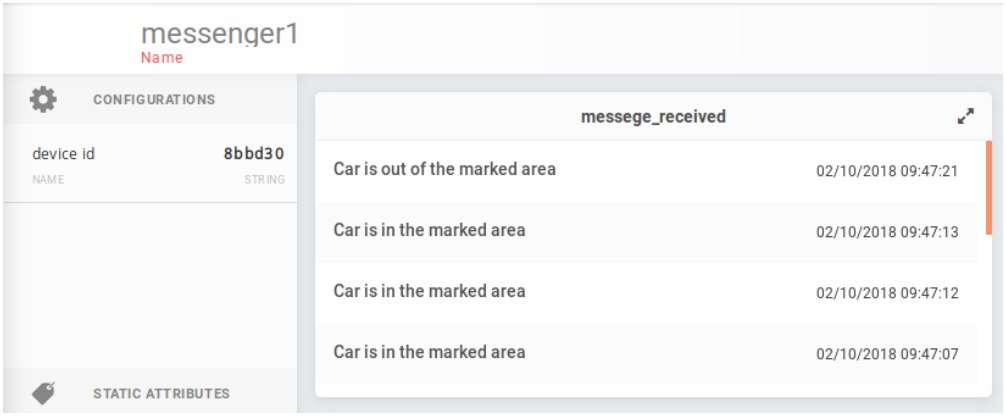


Fig. 11.29: : Saída no dispositivo

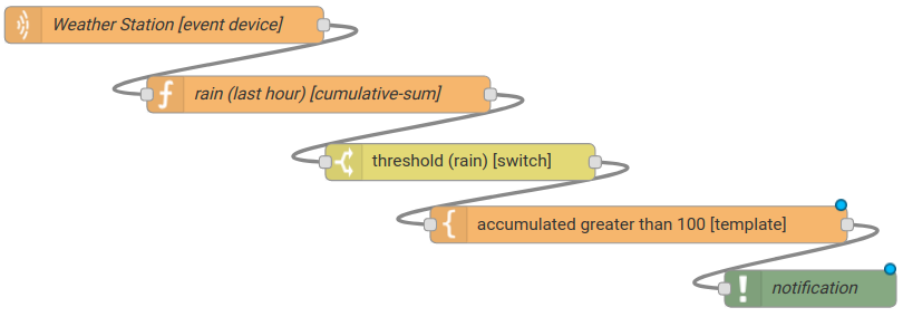


Fig. 11.30: : Fluxo usando os nós de soma acumulada, interruptor and notificação

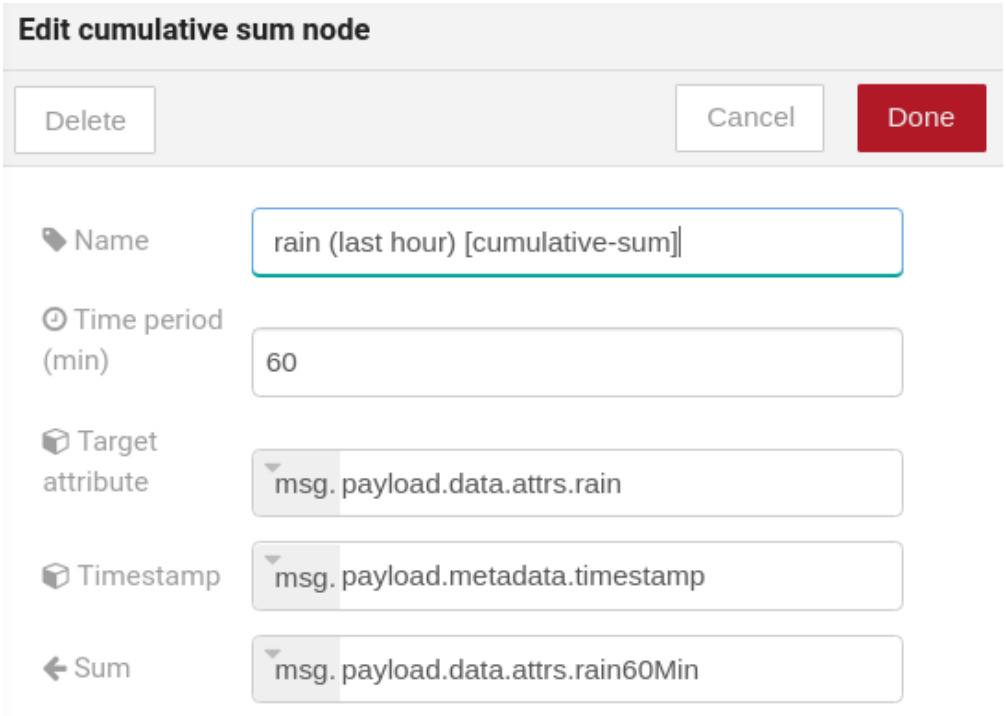


Fig. 11.31: : Configuração de Soma acumulada

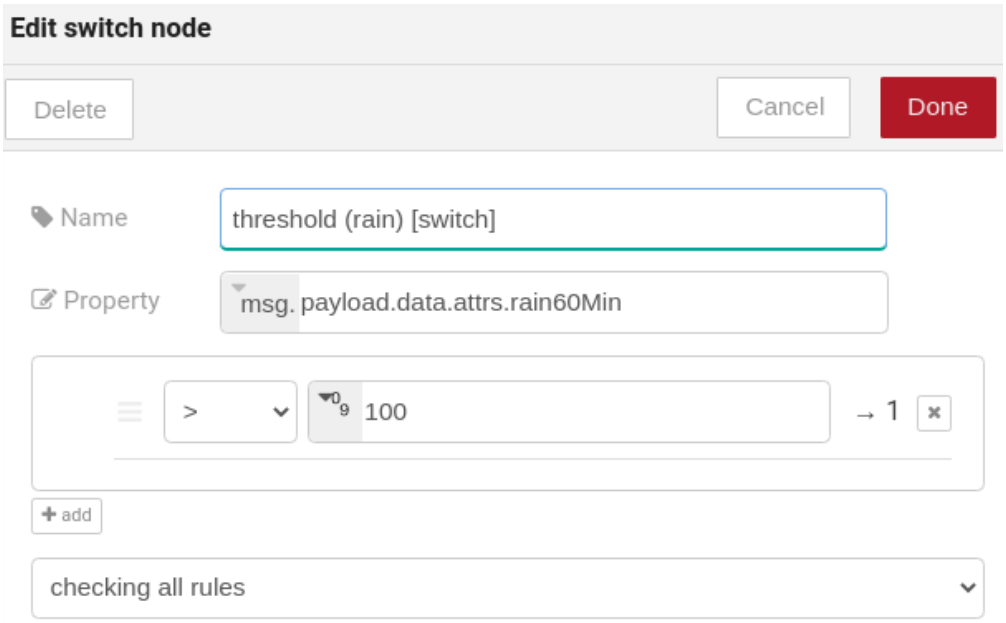


Fig. 11.32: : Configurações do nó interruptor (Switch)

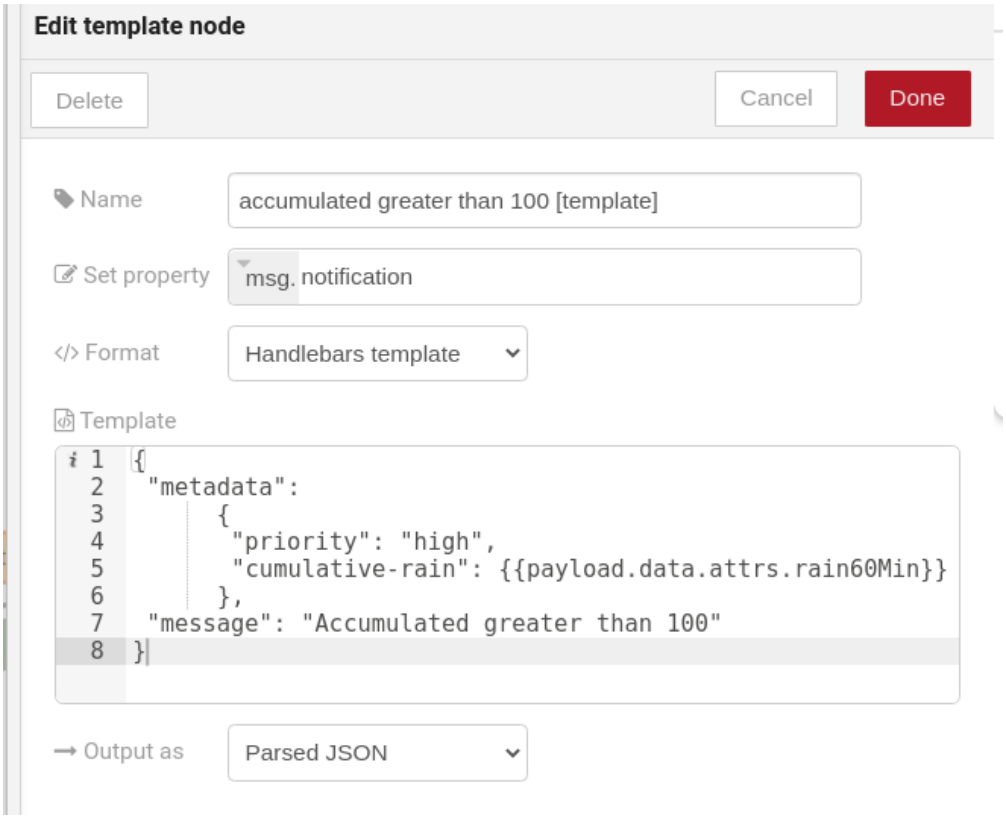


Fig. 11.33: : Configuração do nó do modelo

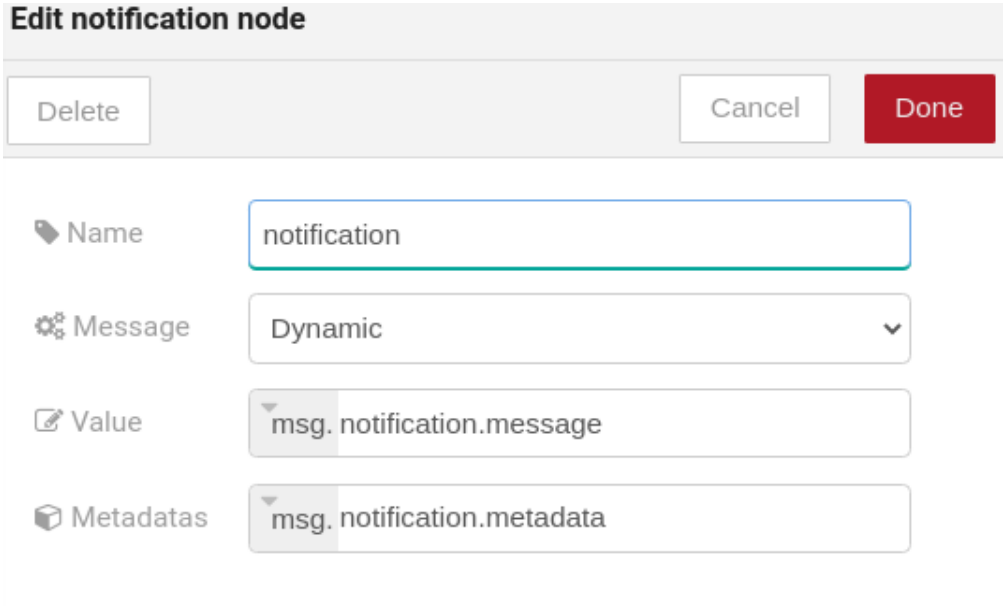


Fig. 11.34: : Configuração do nó de notificação

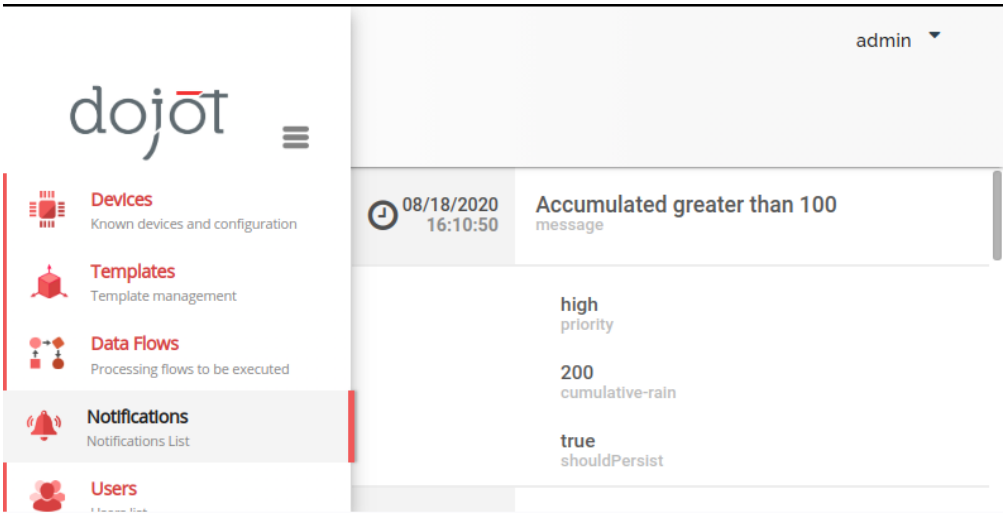


Fig. 11.35: Notificação (notification)

CAPÍTULO 12

Usando MQTT com segurança (TLS)

Nota:

- Público: administradores
- Nível: intermediário
- Tempo de leitura: 15 ms

Este documento descreve como configurar o dojot para usar o MQTT sobre TLS quando usando o microsserviço IotAgent-Mosca

Índice

Para que um dispositivo se conecte usando o TLS com o Mosca, ele deve possuir:

- Um par de chaves (arquivo .key)

- Um certificado assinado por uma Autoridade de Certificação (CA) confiável por Mosca(arquivo .crt);
- O certificado desta CA (arquivo .crt);

Quando um dispositivo é criado, o DeviceManager notifica automaticamente os seguintes componentes:

- IoTAgent-Mosca: registrará o novo dispositivo em seu cache interno e criará uma entrada, permitindo que o dispositivo publique em um tópico especifico.
- EJBCA: criará uma entidade final para que um certificado possa ser criado no futuro.

12.1 Componentes

12.1.1 EJBCA-REST

O EJBCA‘_ é uma infraestrutura de chave privada (PKI) completa capaz de gerenciar CAs, chaves de criptografia e certificados. O EJBCA fornece SOAP, web e uma interface de linha de comandos. O EJBCA-REST é um wrapper no topo do EJBCA que fornece interface moderna como REST .

O EJBCA fornece interfaces SOAP, web e de linha de comando. EJBCA-Rest é um wrapper do EJBCA que o complementa, permitindo que a CA seja configurado usando REST. Quando usado na dojot, ele também escuta eventos Kafka, permitindo sua configuração automática.

O que é um certificado?

Um certificado contém a chave pública de uma entidade (um usuário, dispositivo, website), juntamente com informações sobre essa entidade, sobre a CA que assina o certificado, o uso permitido do certificado e uma soma de verificação. Quando uma entidade deseja que um certificado seja assinado, a entidade deve criar um arquivo CSR e enviá-lo para a CA desejada. O arquivo CSR é uma ‘intenção de certificação’. O arquivo contém as informações necessárias da entidade e algumas informações sobre o uso do certificado, nomes de host e IPs onde o certificado residirá, nomes alternativos para a entidade etc. A EJBCA pode decidir, usando suas políticas configuradas, quais informações manter, descartar e substituir do CSR recebido. EJBCA pode recusar assinar um CSR se concluir que não é seguro o suficiente de acordo com sua políticas.

Essas políticas configuráveis são chamadas de ‘Perfis de certificado’. Um perfil do certificado chamado CFREE, especializado para MQTT TLS, é fornecido.

Em resumo, o CFREE tem as seguintes configurações (e muitas mais):

- As chaves de criptografia devem ter entre 2048 e 8192 bits;
- As entidades podem definir nomes de host e IPs;
- O uso da chave está marcado como não crítico (por enquanto);
- O algoritmo de hash é SHA256. O algoritmo de assinatura é RSA.

Então, como o EJBCA funciona no dojot?

Ao criar um novo dispositivo, uma entidade associada é criada na EJBCA. Seu nome será o ID do dispositivo (como ‘f60c28’) e sua senha será sempre ‘dojot’.

Um certificado pode ser assinado enviando uma solicitação HTTP POST para host:8000/sign/<cname>/pkcs10. CName é o nome da entidade final (ou device). O payload enviada com esta solicitação deve ser um JSON contendo a senha da entidade final e um arquivo CSR (intenção de certificado) em base64.

Observe que a URL é 'roteado' pelo gateway da API. Como em outras APIs na dojot, é necessário um JWT. Você pode descobrir como gerar e como usar tal JWT em *Utilizando a API da dojot*.

Para criar o arquivo CSR e solicitar uma assinatura de certificado, um usuário pode usar um script auxiliar chamado 'Certificate Retriever', que é detalhado na seção *Certificate retriever*.

12.1.2 Mosca

Mosca é um broker mqtt em node.js. Para usar o Mosca, você precisa fazer algumas configurações por variável de ambiente:

- MOSCA_TLS_DNS_LIST: lista de DNSs TLS, o host para conectar externamente (separado por vírgula). Exemplo: localhost, meudominio.com

Todos os certificados serão criados automaticamente no broker, sem a necessidade de configurar manualmente os certificados no broker.

Nota: Para usar Mosca sem TLS também, você precisa definir a variável de ambiente ALLOW_UNSECURED_MODE para 'true' e usar a porta 1883. **Não é recomendado!**

12.2 Certificate retriever

Este componente é um script auxiliar para a criação de certificados de dispositivo. É disponível em Certificate Retriever GitHub repository e codificado usando Python 3.

Um usuário pode usá-lo executando:

```
sudo apt install python3-pip #run on Debian-based Linux distributions, if necessary

pip3 install crypto #or pip install crypto, run if necessary
pip3 install pyOpenSSL #or pip install pyOpenSSL, run if necessary
pip3 install requests #or pip install requests, run if necessary

mkdir -p certs
```

E finalmente obter o certificado para o dispositivo:

```
python3 generateLoginPwd.py  ${DOJOT_HOST} ${DEVICE_ID} IOTmidCA #run every time
```

Os parâmetros obrigatórios são:

- ${DOJOT_HOST}: onde está o dojot (Sem / no final). Exemplo: http://localhost:8000
- ${DEVICE_ID}: ID do dispositivo que receberá um novo certificado. Exemplo: f60c28

Observe que a autenticação é realizada na dojot. O script solicitará credenciais do usuário e invocará a autenticação do usuário automaticamente. O usuário precisa de permissão para assinar o certificado para poder usar essa rota.

Uma entidade final deve existir na EJBCA no estado 'Novo' antes de solicitar uma nova assinatura de certificado. Quando um novo dispositivo é criado, uma entidade final é criado automaticamente no EJBCA pelo DeviceManager. Esta nova entidade finalname é o próprio ID do dispositivo. Sua senha é 'dojot'.

O script autentica os usuários com nome de usuário e senha, recupera o certificado da CA, gera um par de chaves e um arquivo CSR e solicita a assinatura do certificado, nesta ordem. Qualquer erro em qualquer etapa será interrompa sua execução.

Após executado com sucesso, todos os certificados podem ser encontrados na pasta './certs'.

12.3 Simulando um dispositivo com *mosquitto*

Para publicar e assinar usando os certificados apropriados, você deve estar com o Mosca Broker e o EJBCA em execução. Depois de criar o ambiente dojot, os modelos (templates) e os dispositivos. Então usando o *mosquitto* emule um dispositivo, publique e subscreva-se nos tópicos desejados:

Antes de instalar mosquitto_pub e mosquitto_sub (do pacote *mosquitto-clients* em Debian-based Linux distribuições) e acessar a pasta certs, se necessário:

Atenção: Algumas distribuições Linux, distribuições Linux baseadas em Debian em particular, tem dois pacotes para mosquitto - um contendo ferramentas para cliente (ou seja, mosquitto_pub e mosquitto_sub para publicar mensagens eassinando tópicos) e outro contendo o broker MQTT também. E neste tutorial, apenas as ferramentas do pacote *mosquitto-clients* em Distribuições Linux baseadas no Debian serão usadas. Verifique se o broker MQTT não está em execução antes de iniciar o dojot (executando comandos como `ps aux | grep mosquitto`) para evitar conflitos de porta.

```
sudo apt-get install mosquitto-clients   #if necessary on Debian-based Linux␣
↪distributions
cd certs  #if necessary
```

Como publicar:

```
mosquitto_pub  -h localhost -p 8883 -t /<tenant>/<deviceId>/attrs -i <tenant>:
↪<deviceId> -m '{"attr_example": 10}' --cert <your .crt file> --key <your .key file>␣
↪--cafile IOTmidCA.crt
```

Como se subscrever:

```
mosquitto_sub  -h localhost -p 8883 -t /<tenant>/<deviceId>/config -i <tenant>:
↪<deviceId> --cert <your .crt file> --key <your .key file> --cafile IOTmidCA.crt
```

O <seu arquivo .crt>, <seu arquivo .key> e o cafile podem ser criados com o script do Certificate Retriever GitHub repository. Onde <tenant> é um identificador de contexto na dojot e <deviceId> é um identificador para o dispositivo no contexto correspondente.

Nota: neste caso, a mensagem é uma publicação com um atributo, este atributo tem o rótulo *attr_example* e um novo valor 10, você precisará mudar isso para o seu caso de uso.

12.4 Anotações importantes

Estas são algumas notas importantes, relacionadas à segurança do dispositivo e assuntos associados.

12.4.1 Debugging

Os erros do TLS podem não ser tão detalhados quanto outros problemas. Se houver um erro, o usuário pode não saber o que deu errado porque nenhum componente indica algum problema. Nesta seção, existem algumas dicas, frequentemente as ferramentas de depuração ajudam a descobrir o que está acontecendo.

Como ler um certificado

Um arquivo de certificado pode estar em dois formatos: PEM (texto base64) ou DER(binário). O OpenSSL oferece ferramentas para ler esses formatos:

```
openssl x509 -noout -text -in certFile.crt
```