

Middleware Orientado a Mensagem para IoT (aplicação em SmartHome)

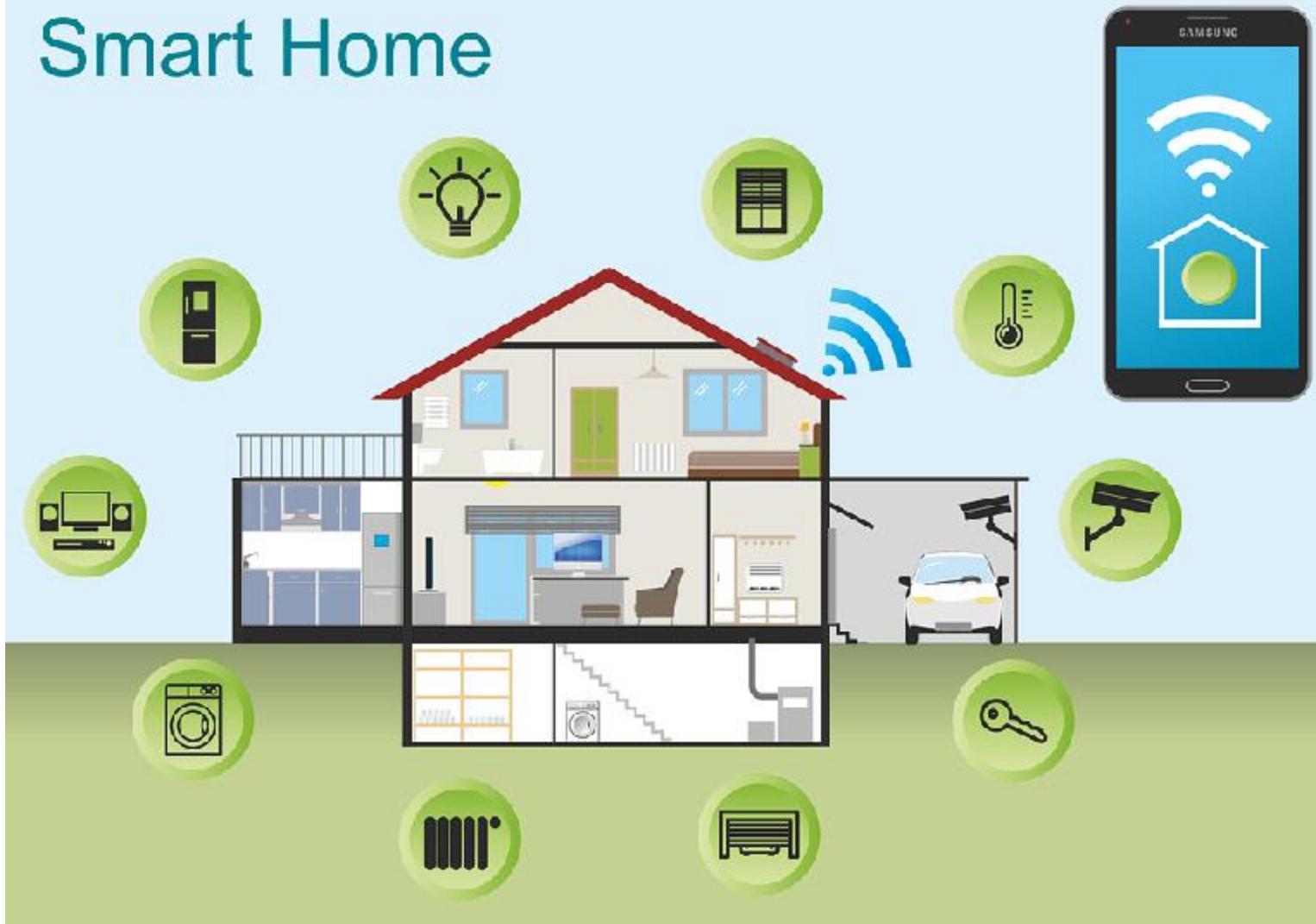
Aluna: Daniele Brooman (dwb)

Objetivo

- O projeto tem como objetivo desenvolver um middleware para aplicações em IoT, em específico projetos de SmartHome;
- Esse tipo de aplicação requer que o servidor se comunique com diversos dispositivos, não sendo necessária sincronicidade de comunicação (sensores e atuadores em dispositivos distintos).
- Dessa forma, foi selecionado o modelo de Middleware Orientado a Mensagem (MOM).

Cenário de uso - SmartHome

Smart Home



Cenário de uso - SmartHome

Smart Home



Background

■ IoT

- Objetos do cotidiano interconectados digitalmente via internet;
- Capazes de coletar e transmitir dados;
- Diferentes tipos de informação;
- Sistemas automatizados;
- Adição de elementos (sensors/atuadores);
- Dispositivos embarcados com comunicação sem fio;

Background

■ MOM

- Comunicação assíncrona;
- Comunicação anônima;
- Gerenciamento de filas
 - Modelos Point-to-Point ou Publisher/Subscriber;
- Comunicação 1-1 ou 1-N;

- Mensagem com prioridade
- Retirar vs manter mensagem na fila

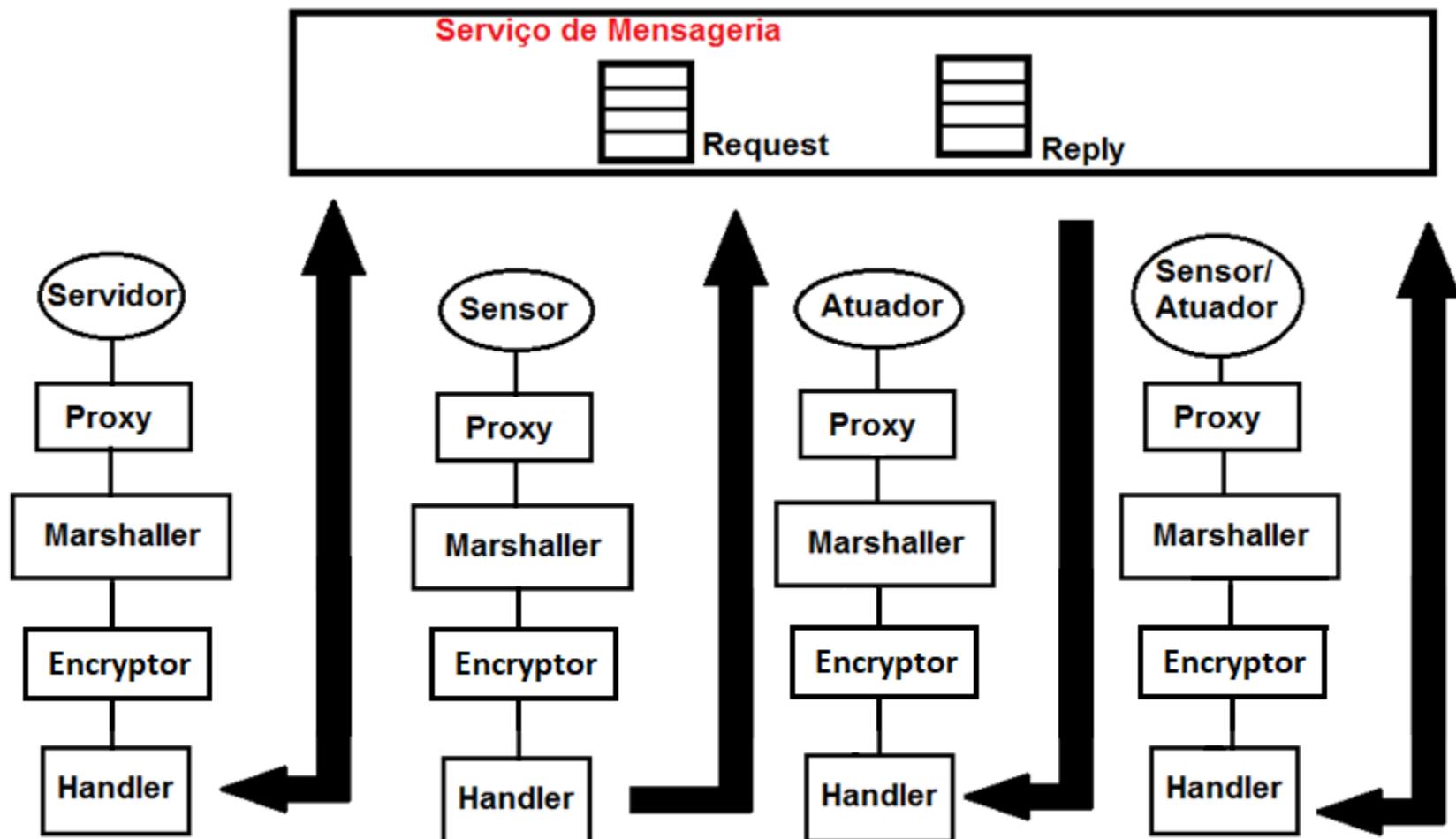
Requisitos

#	Descrição do Requisito Funcional
RF01	Gerenciar filas
RF02	Tratar conexões/desconexões
RF03	Serializar dados
RF04	Criptografar mensagens
RF05	Armazenar informações sobre dispositivos

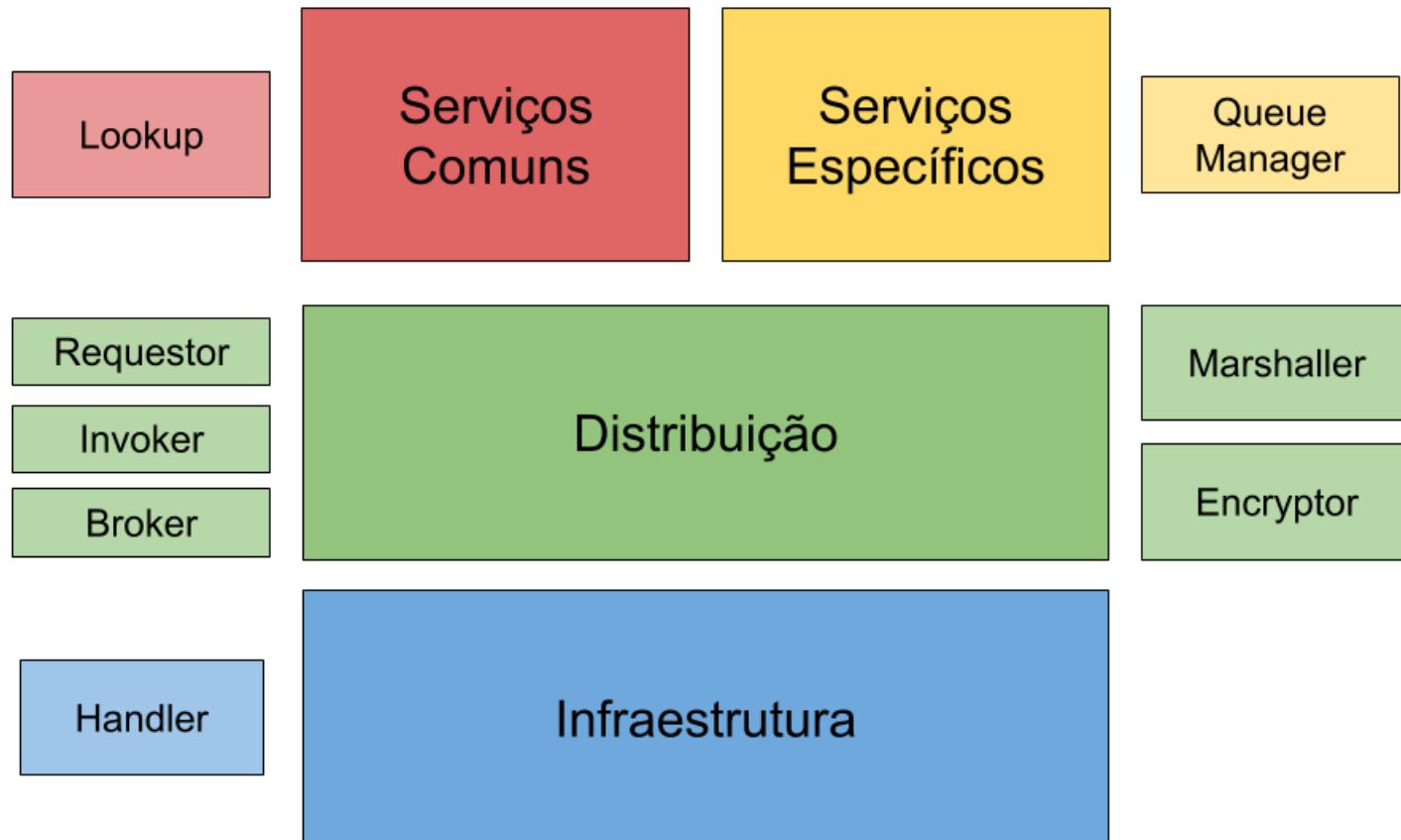
#	Descrição do Requisito Não-Funcional
RNF01	Abstração de heterogeneidade (de tipos de mensagem)
RNF02	Segurança de mensagem

#	Transparências
RT01	Falha de comunicação
RT02	Acesso

Arquitetura



Arquitetura



Arquitetura

- **Gerenciar fila: Queue Manager**
- **Tratar conexões/desconexões: Handler**
- **Serializar dados: Marshaller**
- **Criptografar mensagens: Encryptor**
- **Armazenar informações sobre os dispositivos: Lookup**
- **Gerenciar fluxo de informações: Request, Broker, Invoker**

Projeto

■ Padrões de projeto:

- Marshaller
 - Serialização de dados
- Proxy
 - Transparência de acesso
- Handler
 - Gerenciar conexões
- Requestor, Broker, Invoker
 - Acesso a objeto remoto/gerenciador de fila

■ Padrões de Identificação

- Lookup
 - Armazenamento de dados do dispositivos cadastrados

Implementação

■ Queue Manager

```
type Queue struct {
    name          string
    messages      []string
    messagesCounter int
}
```

```
func (Impl *SMImplementation) QueueDeclare(msg IoTMessage.IoTMessage){
    listExist := Impl.CheckQueue(msg)
    if listExist == false {
        queue := &Queue{
            name: msg.Header.ListName,
            messagesCounter: 0,
        }
        queueList = append(queueList, queue) //adiciona nova fila na lista geral de filas
    }
}
```

Implementação

■ Queue Manager (Consume)

```
func (Impl *SMImplementation) Consume(msg IoTMessage.IoTMessage) string {
    line := "\nil"
    for i := range queueList {
        if msg.Header.ListName == queueList[i].name{
            if queueList[i].messagesCounter > 0 {           //se o contador de mensagens disponíveis estiver maior que 0
                line = queueList[i].messages[0]             //é salva a linha de mensagem na primeira posição do buffer
            }
            break
        }
    }
    return line
}

func (Impl *SMImplementation) Remove(msg IoTMessage.IoTMessage) {
    for i := range queueList {
        if msg.Header.ListName == queueList[i].name{
            if queueList[i].messagesCounter > 0 {           //se o contador de mensagens disponíveis estiver maior que 0
                if msg.Payload.Message == "true" {
                    queueList[i].messages = append(queueList[i].messages[:0], queueList[i].messages[1:]...)
                    queueList[i].messagesCounter--            //o contador de mensagens do cliente é diminuído
                } else {
                    line := queueList[i].messages[0]         //é salva a linha de mensagem na primeira posição do buffer
                    queueList[i].messages = append(queueList[i].messages[:0], queueList[i].messages[1:]...)
                    queueList[i].messages = append(queueList[i].messages, line)
                }
            }
            break
        }
    }
}
```

Implementação

■ Queue Manager (Publish)

```
func (Impl *SMImplementation) Publish(msg IoTMessage.IoTMessage, priority bool) {
    line := string(msg.Payload.Message)

    for i := range queueList {
        if msg.Header.ListName == queueList[i].name{
            if priority == false {
                queueList[i].messages = append(queueList[i].messages, line)          //Caso
            } else {
                queueList[i].messages = append([]string{line}, queueList[i].messages...)
            }

            queueList[i].messagesCounter++
            break
        }
    }
}
```

Implementação

■ Encryptor

- Advanced Encryption Standard (AES)

```
func (Encrypt *Encryptor) Encrypt(data []byte, passphrase string) []byte {
    key := []byte(Encrypt.CreatHash(passphrase))
    block, _ := aes.NewCipher(key)
    gcm, _ := cipher.NewGCM(block)
    nonce := make([]byte, gcm.NonceSize())
    io.ReadFull(rand.Reader, nonce)
    ciphertext := gcm.Seal(nonce, nonce, data, nil)
    return ciphertext
}
```

```
func (Encrypt *Encryptor) Decrypt(data []byte, passphrase string) []byte {
    key := []byte(Encrypt.CreatHash(passphrase))
    block, _ := aes.NewCipher(key)
    gcm, _ := cipher.NewGCM(block)
    nonceSize := gcm.NonceSize()
    nonce, ciphertext := data[:nonceSize], data[nonceSize:]
    plaintext, _ := gcm.Open(nil, nonce, ciphertext, nil)
    return plaintext
}
```

Implementação

■ Encryptor

- Advanced Encryption Standard (AES)

```
func (Encrypt *Encryptor) CreatHash(key string) string {  
    hasher := md5.New()  
    hasher.Write([]byte(key))  
    return hex.EncodeToString(hasher.Sum(nil))  
}
```

Implementação

■ Conexão/Desconexão (Device)

```
func (Req *DeviceRequestor) New(port string, persistent bool) {
    Req.Port = port
    if persistent == true {
        Req.Open()
        Req.PortOpen = true
    }
    Req.Persistent = persistent
}
```

```
func (Req *DeviceRequestor) ReadViaHandler() string{
    message, err := Req.Handler.Read()
    if err != nil {
        if Req.Persistent == true {
            for err != nil {
                Req.Open()
                message, err = Req.Handler.Read()
            }
        }
    }
}
```

Implementação

■ Conexão/Desconexão (Device)

```
err := Req.Handler.Write(cryptedMsg)
if err != nil {
    if Req.Persistent == true {
        for err != nil {
            Req.Open()
            err = Req.Handler.Write(cryptedMsg)
        }
    }
}
```

Implementação

■ Conexão/Desconexão (Device)

```
if Req.Persistent == false {  
    Req.Open()  
}  
  
Req.WriteViaHandler(msgRequest)  
  
if answer == true {  
    ans = Req.ReadViaHandler()  
} else {  
    ans = "ack"  
}  
  
if Req.Persistent == false {  
    Req.Close()  
}
```

Implementação

■ Handler (Serviço de Mensageria)

```
func (Sm *SMRequestHandler) Remove (deviceId int) {  
    Sm.ConnCounter--  
    Sm.Conn = append(Sm.Conn[:deviceId], Sm.Conn[deviceId+1:]...)  
}
```

Avaliação de Desempenho

■ Processador

- Intel Core i5-8250U 1,6GHz (com Turbo Boost up to 3,4GHz)

■ Memória

- 8GB

■ Wifi

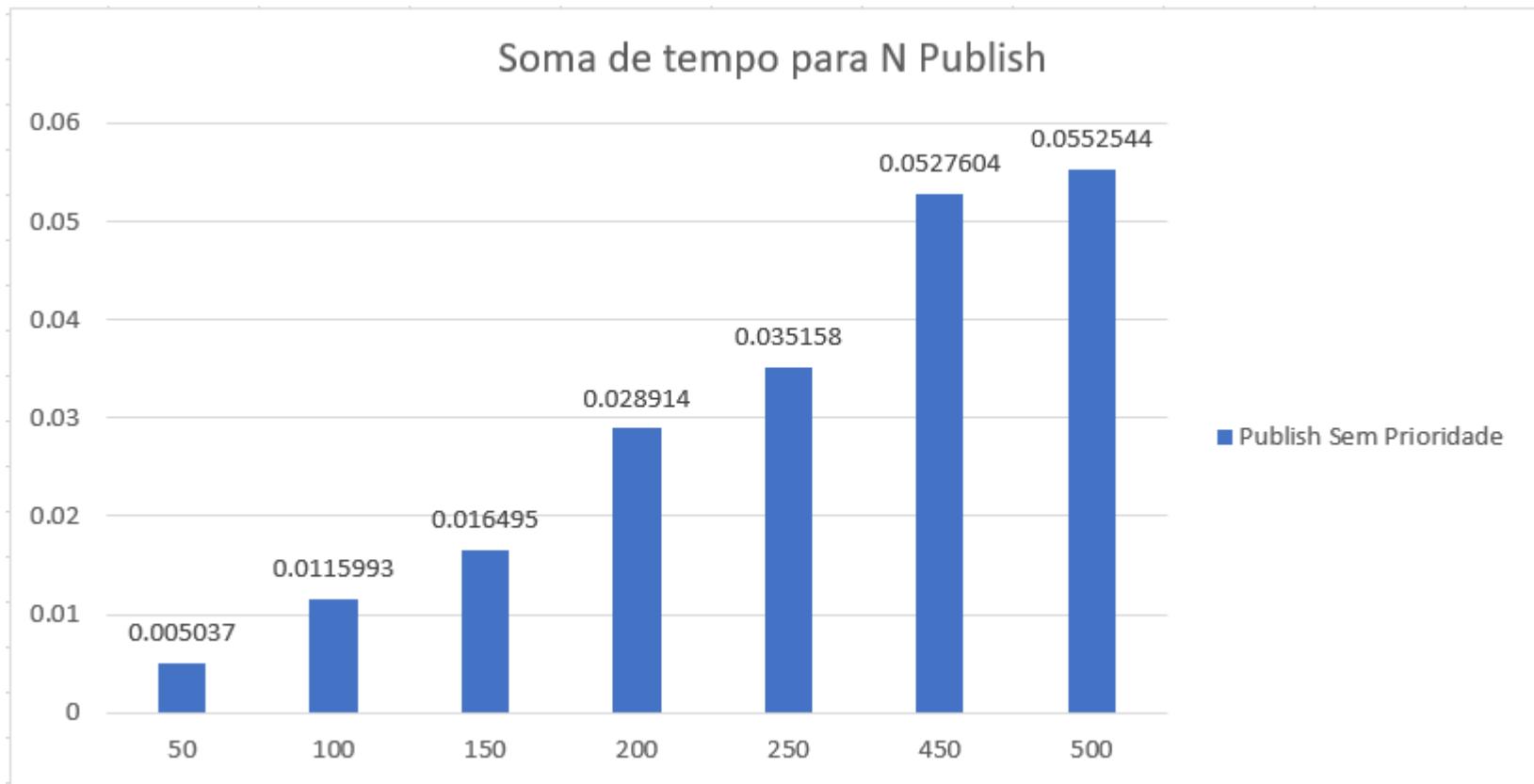
- Desligada

■ Sistema Operacional

- Windows 10 Home (v1803)

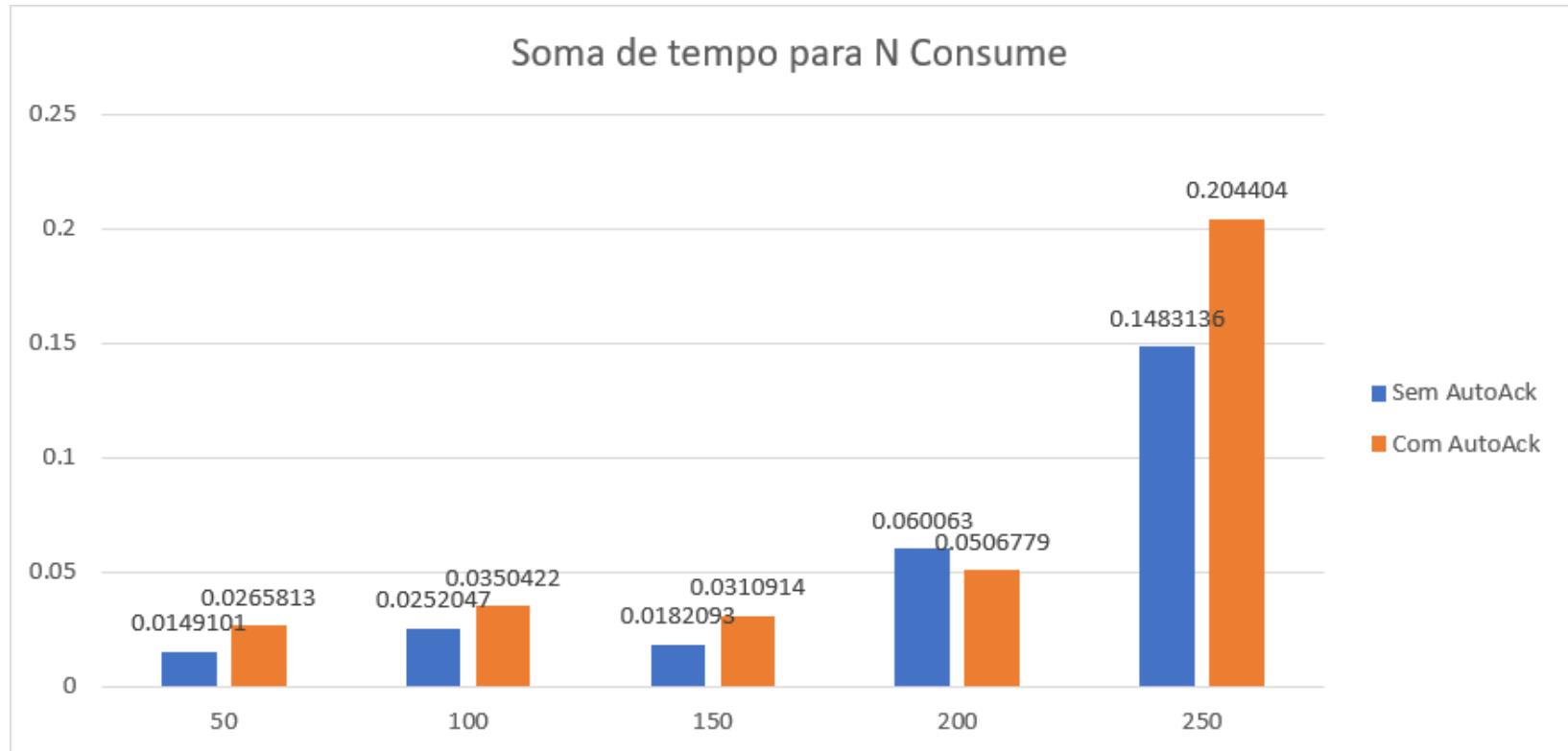
Avaliação de Desempenho

■ Tempo para adicionar mensagem na fila



Avaliação de Desempenho

■ Tempo para ler mensagem da fila



Conclusão

■ Principais características

- Troca de mensagens de maneira assíncrona e anônima
- Conexão persistente ou temporária 1-1 ou 1-N
 - Conexão apenas no momento de transmissão permite um melhor aproveitamento de bateria em embarcados
- Dados criptografados para segurança das informações das aplicações/usuários
- Permite mensagem prioritária
- Possui tanto autoAck quanto Ack manual para remoção de mensagem da fila
 - Caso a mensagem não seja de interesse do dispositivo, ela retorna para o final da fila
- Lookup para armazenar dados dos dispositivos utilizando o serviço de mensageria

Conclusão

■ Limitações

- Não possui mecanismo Publisher/Subscriber
- Limite na frequência de adição/remoção de mensagens na fila
- Não possui heterogeneidade de conexão (conexão apenas por TCP)

Conclusão

■ Lições aprendidas com o projeto

- Definir requisitos (funcionais e não-funcionais)
- Escolher transparências relevantes ao projeto
- Definir elementos de arquitetura de acordo com requisitos
- Aprofundamento dos conhecimentos em MOM e arquitetura de Middlewares
- Introdução a tipos de criptografia indicadas para sistemas de IoT
- Melhor entendimento de testes para avaliação de desempenho

Fim dos Slides