Pushca - lightweight solution for push notifications.

What is Pushca

Pushca is stanalone/clustered solution that facilitates fast and reliable async push notifications delivery via long-live web socket connections. It can manage hundreds of active client connections and forward message from http rest request to all targetted clients via dedicated ws channels.

Pushca was created with scalability in mind and you can easily setup a cluster to keep thousands active clients and provide exceptional throughput with relatively humble resources and open horizontal scaling.

The big advantage of Pushca, compare to other ws based solutions, is ability to send notifications via simple http rest endpoint. So client can receive notifications from any system that supports http post requests and no need to establish permanent ws connection for every sender.

Websockets

WebSockets are a communication protocol that provides full-duplex, bidirectional communication channels over a single TCP connection. Unlike traditional HTTP requests, where the client sends a request and waits for the server to respond, WebSockets allow both the client and the server to initiate communication independently.

WebSockets are preferable in the following scenarios:

- Real-time Interactivity: WebSockets are designed for real-time, interactive communication. If your application requires instant updates and bidirectional communication between clients and servers, WebSockets are a better choice. Examples include live chats, online gaming, collaborative tools, and live notifications.

- Reduced Latency: WebSockets offer significantly lower latency compared to HTTP/REST, as there's no need to establish a new connection for every request. This makes them ideal for scenarios where quick data exchange is critical.
- Server Push: With WebSockets, the server can push updates to clients without clients needing to request them. This is advantageous for applications that involve sending data updates, such as live data feeds or real-time monitoring.
- Efficient Resource Usage: WebSockets use a single, long-lived connection for communication, reducing the overhead associated with setting up and tearing down connections for each request. This can be more efficient when dealing with frequent updates.
- Continuous Streaming: If your application involves streaming data, such as live video or audio, WebSockets can handle the constant data flow without the need for repeated requests.

Pushca under the hood

Pushca is written in java/spring boot 3. For distribution is used native executable binary assembled with GraalVM native image plagin.
Docker image is based on ubuntu:rolling.

Current limitations and possible extensions

Pushca is a very lightweight solution with exceptional throughput and performance. It is not an enterprise monster like Kafka. Lets list features that currently are not supported but can be added in the future:

- Integration with Kafka via topic consumer.
- Repeat logic for failed deliveries.
- Message storage and message history.
- Security for public http REST endpoints.

Currently all that can be achieved with some intermediate services. For instance to address security you can put Pushca behind some secured API gateway on your production environment.

Also important to remember that all re-connect logic especially in environment with unstable network should be implemented on client side.

Pushca claster: components and configuration

Effective production setup usually consists of three components:

1. Pusher - main service that is responsible for managing of web socket (ws) connections and actual sending of notifications.
2. Redis(7.0.2 or higer) - dynamic storage for clients to pusher mapping, service discovery and internal communication.
3. Nginx or Traefik - load balancer and reverse proxy.

Standalone setup requires only first one, cluster - all three mentioned above.

Pusher

Exposes open port for ws connections and also two http rest endpoints:

1. `"/open-connection"` facilitate a new connection creation;
2. `"/send-notification"` send message to connected client.

One Pusher can support up to 1000 connected clients (depends on service resources: memory, CPU etc.)
All configuration can be done via system variables and config file.

Minimal configuration for setup with one pusher.

All parameters documented in the following format:
<env variable>/<config file property>/<+ required, - optional>/<default value>. Env variables override values from config file.

Config file example application.yaml
https://github.com/MaksimBugay/pushca-public/blob/master/docker-swarm/conf/application.yaml

---

```
management:
  endpoint.health.show-details: always
  endpoints.web.exposure:
      include: '*'
      exclude: 'shutdown'

pushca:
  coordinator:
    secret-key: <your secret key for JWT signature>
  pusher:
    deploy-mode: cluster
  license-key: <your license key>

server:
  address: 0.0.0.0
  tomcat:
    threads:
      max: 1000
      min-spare: 500

springdoc:
  packages-to-scan: bmv.org.pushca.api.controller.external
  api-docs:
    path: /pushca
  swagger-ui:
    url: /pushca.yaml
```

The only important customisable parameter is pushca.coordinator.secret-key that is used for JWT signature. Thre rest is commot spring boot application properties for monitoring.

Other important parameters:

- **PUSHCA_SERVER_SOCKET_PORT/pushca.pusher.socket-port/+/8883** Published socket port
- **PUSHCA_EXTERNAL_ADVERTISED_URL/pushca.pusher.external-advertised-url/+/-** Url for ws connection from outside of docker.
- **PUSHCA_INTERNAL_ADVERTISED_URL/pushca.pusher.internal-advertised-url/-/-** Url for ws connection inside docker.
- **PUSHCA_ACKNOWLEDGE_TIMEOUT_MSpushca.acknowledge-timeout.ms/+/1000** acknowledge message timeout
- **spring.config.location/-/+/-** Config file location.

Extended configuration for cluster setup with several pushers.

- **PUSHCA_INTERNAL_CLUSTER_URL/pushca.pusher.internal-cluster-url/+/-** Url for internal http rest communication between pushers.
- **PUSHCA_REDIS_HOST_NAME/pushca.storage.redis.host/+/127.0.0.1** Redis host name.
- **PUSHCA_REDIS_PORT/pushca.storage.redis.port/+/6379** Redis port.

Performance and throughput (all optional).

- **PUSHCA_JEDIS_POOL_SIZE/pushca.storage.redis.pool-size/-/800** Size of redis connections pool.
- **PUSHCA_SERVER_SOCKET_POOL_SIZE/pushca.pusher.socket-pool-size/-/500** Size of ws connections pool (how many clients can be connected simultaneously).
- **PUSHCA_CACHE_SEARCH_TTL_MS/pushca.coordinator.cache.search.ttl-ms/-/5 sec** Local cache for client mapping search results time to live (ms). That parameter speed up message delivery but after re-connect client start receiving messages with some delay.
- **PUSHCA_SERVER_ASYNC_WORKERS_POOL_SIZE/pushca.pusher.async-workers.pool-size/-/1000** Size of async workers thread pool (async message processing inside pusher).

- **-/server.tomcat.threads.max/-/1000** Max number of tomcat threads to serve http rest requests.

Pusher setup example for docker swarm.

https://github.com/MaksimBugay/pushca-public/blob/master/docker-swarm/pusher-backend.yml

---

```
version: '3.8'

services:

 pusher1:
   restart: unless-stopped
   image: n7fr846yfa6ohlhe/mbugai:pushca-clastered-1.0
   environment:
     - 'PUSHCA_REDIS_HOST_NAME=pushca-cache'
     - 'PUSHCA_REDIS_PORT=6379'
     - 'PUSHCA_SERVER_SOCKET_PORT=8885'
     - 'PUSHCA_INTERNAL_ADVERTISED_URL=ws://pusher1:8885/'
     - 'PUSHCA_INTERNAL_CLUSTER_URL=http://pusher1:8080'
     - 'PUSHCA_EXTERNAL_ADVERTISED_URL=ws://82.147.191.51:35085/'
     - 'spring.config.location=/conf/application.yaml'
   configs:
     - source: pushca-config
       target: /conf/application.yaml
   networks:
     default:
       aliases:
         - pusher1-stage
   logging:
     driver: "json-file"
     options:
       max-size: "1m"
       max-file: "10"
   labels:
     filebeats_log: "false"
   deploy:
     labels:
       - traefik.enable=false
     replicas: 1
     placement:
       constraints: [ node.labels.pusher1==true ]
     resources:
       limits:
         cpus: '1.75'
         memory: 8G
       reservations:
```

```
        cpus: '1'
        memory: 4G
    restart_policy:
      condition: any
      delay: 5s
      max_attempts: 9999999
      window: 90s

.
.
.

 pusherN:
   restart: unless-stopped
   image: n7fr846yfa6ohlhe/mbugai:pushca-clastered-1.0
   environment:
     - 'PUSHCA_REDIS_HOST_NAME=pushca-cache'
     - 'PUSHCA_REDIS_PORT=6379'
     - 'PUSHCA_SERVER_SOCKET_PORT=8889'
     - 'PUSHCA_INTERNAL_ADVERTISED_URL=ws://pusherN:8889/'
     - 'PUSHCA_INTERNAL_CLUSTER_URL=http://pusherN:8080'
     - 'PUSHCA_EXTERNAL_ADVERTISED_URL=ws://82.147.191.51:35089/'
     - 'spring.config.location=/conf/application.yaml'
   configs:
     - source: pushca-config
       target: /conf/application.yaml
   networks:
     default:
       aliases:
         - pusherN-stage
   logging:
     driver: "json-file"
     options:
       max-size: "1m"
       max-file: "10"
   labels:
     filebeats_log: "false"
   deploy:
     labels:
       - traefik.enable=false
     replicas: 1
     resources:
       limits:
         cpus: '1.75'
         memory: 8G
       reservations:
         cpus: '1'
         memory: 4G
     restart_policy:
```

```
      condition: any
      delay: 5s
      max_attempts: 9999999
      window: 90s

networks:
 default:
   external: true
   name: "mla-servers-overlay"

configs:
 pushca-config:
   name: pushca-config-${CONFIG_VERSION:-0}
   file: conf/application.yaml
```

Nginx config example for cluster setup.

https://github.com/MaksimBugay/pushca-public/blob/master/docker-swarm/nginx-pusher.yml

---

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
   worker_connections  1024;
}


http {
   map $http_upgrade $connection_upgrade {
       default upgrade;
       '' close;
   }

   upstream pushCluster {
     server pusher1:8080;
.
.
.

     server pusherN:8080;
   }

   upstream websocketPusher1 {
       server pusher1:8885;
   }

.
.
.

   upstream websocketPusherN {
       server pusherN:8889;
   }

   server {
       listen 8885;
       server_name pusher1_websocket;
       location / {
```

```
            proxy_pass http://websocketPusher1;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
            proxy_set_header Host $host;

            proxy_connect_timeout 7d;
            proxy_send_timeout 7d;
            proxy_read_timeout 7d;
        }
    }

.
.
.

    server {
        listen 8889;
        server_name pusherN_websocket;
        location / {
            proxy_pass http://websocketPusherN;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
            proxy_set_header Host $host;

            proxy_connect_timeout 7d;
            proxy_send_timeout 7d;
            proxy_read_timeout 7d;
        }
    }

    server {
      listen 8080;
      server_name pusher_proxy;
      location /pushca/open-connection {
        proxy_pass http://pushCluster/open-connection;

        #standard proxy settings
        proxy_set_header X-Real-IP $remote_addr;
        proxy_redirect off;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-NginX-Proxy true;
        proxy_connect_timeout 30;
        proxy_send_timeout 30;
        proxy_read_timeout 30;
        send_timeout 60;
```

```
    }
    location /pushca/send-notification {
      proxy_pass http://pushCluster/send-notification;

      #standard proxy settings
      proxy_set_header X-Real-IP $remote_addr;
      proxy_redirect off;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header Host $http_host;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header X-NginX-Proxy true;
      proxy_connect_timeout 30;
      proxy_send_timeout 30;
      proxy_read_timeout 30;
      send_timeout 60;
    }

  }

  include       /etc/nginx/mime.types;
  default_type  application/octet-stream;

  log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"
"$http_accpmp_shard_name"';

  access_log  /var/log/nginx/access.log  main;

  sendfile        on;
  #tcp_nopush     on;

  keepalive_timeout  65;

  #gzip  on;

  include /etc/nginx/conf.d/*.conf;
}
```

Pushca claster: establish client connection

As a preliminary step client should request connection string via “/open-connection” POST endpoint.

Inside request client should provide unique object with four required dimensions:

1. Workspace id
2. Account id
3. Device id
4. Application id.

Every dimension can be used later in multicast messages.

Request body example:

```
{
	"Client":{
		"workSpaceId":"workSpaceMain",
		"accountId":"client1@test.ee",
		"deviceId":"311aae05-bade-48bf-b390-47a93a66c89e",
		"applicationId":"MY_APPLICATION"
		}
}
```

Connection string has the following format.

```
ws://{host}:{open ws port}/{jwt token}
```

Connection string will be expired in 30 minutes.

Establish connection with wscat bash script example.

https://github.com/MaksimBugay/pushca-public/blob/master/client/wscat/start-client.sh

---

```
#!/bin/bash
WORKSPACE_ID="workSpaceMain"
ACCOUNT_ID="client1@test.ee"
DEVICE_ID="311aae05-bade-48bf-b390-47a93a66c89e"
APPLICATION_ID="MY_APPLICATION"

REQUEST_BODY=$( jq -n \
                 --arg workSpaceId "$WORKSPACE_ID" \
                 --arg accountId "$ACCOUNT_ID" \
                 --arg deviceId "$DEVICE_ID" \
                 --arg applicationId "$APPLICATION_ID" \
                 '{client:{workSpaceId: $workSpaceId, accountId: $accountId,
deviceId: $deviceId, applicationId: $applicationId}}' )

echo "${REQUEST_BODY}"

wsurl=$(curl -s http://82.147.191.51:8050/open-connection \
 -H 'Content-Type: application/json' \
 -d "$REQUEST_BODY" | jq '.externalAdvertisedUrl')

echo "${wsurl:1: -1}"

wscat -c "${wsurl:1: -1}"
```

Pushca claster: send simple notification

To send message client should send request to http rest POST endpoint “/send-notification” or message via established ws channel.

Request body example:

```
{
	"filter":{
		"workSpaceId":"workSpaceMain",
		"accountId":"client1@test.ee",
		"deviceId":null,
		"applicationId":"MY_APPLICATION",
		"findAny":false,
		"exclude":null
		},
	"message":"Hello dear users!"
}
```

If any field is omitted then message become multicast and will be delivered to all clients with the same defined dimensions.

You can also set the 'findAny' flag to true. In this case, the message will be delivered to a single client that matches the filter, using a round-robin rotation for every new message. With that functionality very easy to have a load balancer for several connected clients with the same application id.

It is possible to specify a list of clients that should be excluded for delivery. Common scenario: do not deliver multicast message to sender.

Request body with exclude example:

```
{
   "filter":{
      "applicationId":"MLA_JAVA_HEADLESS",
      "findAny":false,
      "exclude":[
        {
         "workSpaceId":"workSpaceMain",
         "accountId":"client2@test.ee",
         "deviceId":"baafbc81-5d70-4bb2-a7ab-ce24a0daf35d",
         "applicationId":"MLA_JAVA_HEADLESS"
        }
       ]
     },
   "message":"message1WithExclude"
   }
```

To send message via ws channel you should wrap request as a command.

Json string as a payload example:

```
{
   "command":"SEND_MESSAGE",
   "metaData":{
      "filter":{
         "workSpaceId":"workSpaceAnyTest",
         "applicationId":"MLA_JAVA_HEADLESS",
         "findAny":false,
         "exclude":[
           {
            "workSpaceId":"workSpaceAnyTest",
            "accountId":"client1@test.ee",
            "deviceId":"8ea6e257-3904-4b5e-8a3a-465e03c7a06c",
            "applicationId":"MLA_JAVA_HEADLESS"
           }
         ]
      },
      "message":"excludeSenderTest"
   }
}
```

Pushca claster: send notification with acknowledge

If message delivery guaranty is important then “/send-notification-with-acknowledge” endpoint or “SEND_MESSAGE_WITH_ACKNOWLEDGE” command should be used. Only direct message can be acknowledge (not multicast). So the request body is a bit different.

Request body example:

```
{
  "id": "messageId",
  "client": {
    "workSpaceId": "workSpaceMain",
    "accountId": "client1@test.ee",
    "deviceId": "8ea6e257-3904-4b5e-8a3a-465e03c7a06c",
    "applicationId": "JAVA_HEADLESS"
  },
  "message": "messageWithAcknowledgeTest"
}
```

Command json example:

```
{
   "command":"SEND_MESSAGE_WITH_ACKNOWLEDGE",
   "metaData":{
      "client":{
         "workSpaceId":"workSpaceAcknowledgeTest",
         "accountId":"client2@test.ee",
         "deviceId":"982f7268-06fb-4f48-8804-bac46abbaebd",
         "applicationId":"MLA_JAVA_HEADLESS"
      },
      "id":"d3259cf4-e41c-4e83-b751-514f42b7e3e2",
      "message":"WithAcknowledgeTest1"
   }
}
```

Very important to remember that acknowledge is a responsibility of client. Ws client should acknowledge every message like “<message id>@@<message body>” with dedicated ACKNOWLEDGE command via ws channel.

Message example:

```
018a842a-7e52-7692-a344-0d76460e1f3f-0@@WithAcknowledgeTest1
```

ACKNOWLEDGE command body example:

```
{
   "command":"ACKNOWLEDGE",
   "metaData":{
      "messageId":"018a8430-cd74-720d-8ff3-083af68d0760-0"
   }
}
```

To track delivery via ws channel effectively client can provide its own message id with “id” request metadata field. For REST endpoint 200 response already stands for successful acknowledge.

OpenAPI specification

https://github.com/MaksimBugay/pushca-public/blob/master/open-api/pushca.yaml

---

```yaml
openapi: 3.0.1
info:
  title: OpenAPI definition
  version: v0
servers:
- url: http://localhost:8050
  description: Generated server url
paths:
  /send-notification:
    post:
      tags:
      - public-api-controller
      summary: Send notification
      operationId: sendNotification
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SendNotificationRequest'
        required: true
      responses:
        "200":
          description: Success
        "429":
          description: Too many requests
  /send-notification-with-acknowledge:
    post:
      tags:
      - public-api-controller
      summary: Send notification with acknowledge
      operationId: sendNotificationWithAcknowledge
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SendNotificationWithAcknowledgeRequest'
        required: true
      responses:
        "408":
          description: Acknowledge time out
        "200":
          description: Success
```

```yaml
  /open-connection:
    post:
      tags:
      - public-api-controller
      summary: Open connection
      operationId: openConnection
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/OpenConnectionRequest'
        required: true
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/OpenConnectionResponse'
        "429":
          description: Too many requests
components:
  schemas:
    ClientSearchData:
      type: object
      properties:
        workSpaceId:
          type: string
        accountId:
          type: string
        deviceId:
          type: string
        applicationId:
          type: string
        findAny:
          type: boolean
        exclude:
          type: array
          items:
            $ref: '#/components/schemas/PClient'
    PClient:
      type: object
      properties:
        workSpaceId:
          type: string
        accountId:
          type: string
        deviceId:
          type: string
```

```
    applicationId:
      type: string
SendNotificationRequest:
  type: object
  properties:
    id:
      type: string
    filter:
      $ref: '#/components/schemas/ClientSearchData'
    message:
      type: string
SendNotificationWithAcknowledgeRequest:
  type: object
  properties:
    id:
      type: string
    client:
      $ref: '#/components/schemas/PClient'
    message:
      type: string
OpenConnectionRequest:
  type: object
  properties:
    client:
      $ref: '#/components/schemas/PClient'
OpenConnectionResponse:
  type: object
  properties:
    externalAdvertisedUrl:
      type: string
    internalAdvertisedUrl:
      type: string
```

Pushca claster: load tests results

Scenario: Iinitialize a pool of connected clients. Pickup two clients randomly and send a message from client1 to client2. Verify delivery on client2 side.

==========

Single mode

==========

ws channel

```
Number of active clients
 1000

Number of successful deliveries during three minute
 520132

Average number of requests per minute
 173377

Number of failed deliveries
 0

Delivery time
 25th pct 6 ms
 50th pct 11 ms
 75th pct 26 ms
 99th pct 184 ms
```

----------h

ttp channel

```
Number of active clients
 800

Number of successful deliveries during three minutes
 59324

Average number of requests per minute
 19774

Number of failed deliveries
 0

Delivery time
 25th pct 3 ms
 50th pct 5 ms
 75th pct 12 ms
 99th pct 93 ms
```

==========

Cluster mode

==========

ws channel

Number of active clients
 1000

Number of successful deliveries during three minutes
 2 313 610

Average number of requests per minute
 771 203

Number of failed deliveries
 0

Delivery time
 25th pct 2 ms
 50th pct 14 ms
 75th pct 57 ms
 99th pct 228 ms

---

http channel

Number of active clients
 1000

Number of successful deliveries during five minutes
 993270

Number of failed deliveries
 0

Average number of requests per minute
 198654

Delivery time
 25th pct 3 ms
 50th pct 4 ms
 75th pct 9 ms
 99th pct 97 ms

Pushca cluster setup

1. 7 Pusher instances

```
resources:
  limits:
    cpus: '1.75'
    memory: 8G
  reservations:
    cpus: '1'
    memory: 4G
```

2. 3 Nginx load balancer instances

```
resources:
  limits:
    cpus: '1'
    memory: 1500M
  reservations:
    cpus: '0.5'
    memory: 800M
```

3. 1 Redis instance