EBook Gratis

APRENDIZAJE PubNub

Free unaffiliated eBook created from
Stack Overflow contributors.

#pubnub

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: pubnub

It is an unofficial and free PubNub ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official PubNub.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con PubNub

Observaciones

Este es un ejemplo simple, pero completo, de inicializar PubNub, suscribirse a un canal y publicar en ese canal.

- Una vez que inicie PUBNUB, puede suscribirse a un canal.
- La devolución de llamada de `connect` indica que la suscripción al canal fue exitosa, por lo que llamamos a nuestra función de `pub` que realiza una `publish` en el canal al que nos suscribimos.
- Este mensaje publicado se enviará a la red de PubNub que enviará el mensaje a todos los suscriptores activos. En este caso, solo estamos nosotros, por lo que recibiremos ese mensaje en nuestra devolución de llamada del `message` donde mostraremos los diversos atributos del mensaje recibido en la Consola de nuestro navegador.

En un caso de uso del mundo real, actualizaría la interfaz de usuario de su página web para mostrar el mensaje recibido.

Ver también: último / oficial PubNub JavaScript SDK Docs

Versiones

Versión	Fecha de lanzamiento
3.15.x	2016-04-01

Examples

Publicar en Subscribe Success (conectar)

Este ejemplo muestra cómo suscribirse, y una vez que tiene éxito, publicar un mensaje en ese canal. También demuestra el conjunto completo de parámetros que se pueden incluir en la función de devolución de llamada del `message` del `subscribe` .

```
pubnub = PUBNUB({
    publish_key   : 'your_pub_key',
    subscribe_key : 'your_sub_key'
});

pubnub.subscribe({
    channel : "channel-1",
    message : function (message, envelope, channelOrGroup, time, channel) {
        console.log(
        "Message Received." + "\n" +
        "Channel or Group: " + JSON.stringify(channelOrGroup) + "\n" +
```

```
        "Channel: " + JSON.stringify(channel) + "\n" +
        "Message: " + JSON.stringify(message) + "\n" +
        "Time: " + time + "\n" +
        "Raw Envelope: " + JSON.stringify(envelope)
    )},
    connect:    pub,
    disconnect: function(m) {console.log("DISCONNECT: " + m)},
    reconnect:  function(m) {console.log("RECONNECT: " + m)},
    error:      function(m) {console.log("ERROR: " + m)}
});

function pub() {
   pubnub.publish({
        channel : "channel-1",
        message : {"msg": "I'm Puuumped!"},
        callback: function(m) {console.log("Publish SUCCESS: " + m)},
        error: function(m) {console.log("Publish ERROR: " + m)}
   })
};
```

Lea Empezando con PubNub en línea: https://riptutorial.com/es/pubnub/topic/327/empezando-con-pubnub

Capítulo 2: Almacenamiento y reproducción

Parámetros

Parámetro	`Type` / `Required` / `Default` Descripción por `Type` / `Required` / `Default`
canal	`String` / `Yes` Especifica el canal para devolver los mensajes del historial.
marcha atrás	`Boolean` / `No` / `false` configuración a verdadero atravesará la línea de tiempo en sentido inverso comenzando con el mensaje más antiguo primero. El valor predeterminado es falso. Si se proporcionan los argumentos de inicio y finalización, se ignora la inversión y se devuelven los mensajes comenzando con el mensaje más reciente.
límite	`Number` / `No` / `100` Especifica el número de mensajes históricos para devolver.
comienzo	`Number` / `No` token de tiempo que delimita el inicio del segmento de tiempo (exclusivo) para extraer mensajes.
fin	`Number` / `No` token de tiempo que delimita el intervalo de fin de tiempo (inclusive) para extraer mensajes.
incluir timetoken	`Boolean` / `No` / `false` Si es verdadero, las marcas de tiempo de publicación del mensaje se incluirán en la respuesta del historial.

Observaciones

Simplemente habilitando el complemento de *almacenamiento y reproducción* para sus claves en el Tablero de administración de PubNub, se almacenarán todos los mensajes publicados en todos los canales. Puede evitar que un mensaje se almacene pasando el parámetro `storeInHistory` como `false` cuando se publica el mensaje, de esta manera:

```
pubnub.publish(
    {
        message: {
            'price': 8.07
        },
        channel: 'channel1',
        storeInHistory: false // override default storage options
    },
    function (status, response) {
        // log status & response to browser console
        console.log("STATUS  : " + console.log(JSON.stringify(status));
        console.log("RESPONSE: " + console.log(JSON.stringify(response));
    }
);
```

De lo contrario, simplemente omita `storeInHistory` o establezca en `true` para almacenar el

mensaje cuando se publique.

Examples

JavaScript SDK v4 - Historia

```
// initialize pubnub object
var pubnub = new PubNub({
    subscribeKey: "yourSubscribeKey",
    publishKey: "myPublishKey" // optional
})

// get up to the last 100 messages
// published to the channel
pubnub.history(
    {
        channel: 'channel1'
    },
    function (status, response) {
        // log status & response to browser console
        console.log("STATUS  : " + console.log(JSON.stringify(status));
        console.log("RESPONSE: " + console.log(JSON.stringify(response));
    }
);

// this is the format of the response
[
    [array of returned messages],
    "firstMessageTimetoken",
    "lastMessageTimetoken"
]

// example of response
[
    [{'price':10.02}, {'price':10.12}, {'price':10.08}, {'price':10.10}],
    "14691304969408991",
    "14691307326690522"
]
```

Lea Almacenamiento y reproducción en línea:
https://riptutorial.com/es/pubnub/topic/3714/almacenamiento-y-reproduccion

Capítulo 3: Controlador de flujo: Grupos de canales

Observaciones

Cuando utilice Grupos de canales, no debe agregar ni eliminar canales en las aplicaciones del lado del cliente. Este ejemplo muestra cómo agregar canales a un grupo de canales y cómo suscribirse a ese grupo de canales por motivos de simplicidad. Pero en un escenario del mundo real, debe hacer que su servidor realice todas las acciones de agregar / eliminar canales a / desde grupos de canales. Cuando habilite Access Manager, necesitará el permiso de `manage` para agregar / eliminar canales a / de los grupos de canales y nunca debe otorgar el permiso de `manage` a los clientes por razones de seguridad. Solo a su servidor se le debe otorgar el permiso de `manage` .

Examples

Suscríbete con grupos de canales en JavaScript

Con el complemento Stream Controller habilitado, puede usar grupos de canales para suscribirse a miles de canales desde un solo cliente. Esto se hace creando un grupo de canales y agregando canales al grupo de canales. `pubnub` variable `pubnub` se ha inicializado correctamente con sus claves.

Crear una función de controlador de devolución de llamada genérica:

```
function displayCallback(m,e,c,d,f){
    console.log(JSON.stringify(m, null, 4));
}
```

Crea un grupo de canales y agrégale canales:

```
pubnub.channel_group_add_channel({
  callback: displayCallback,
  error: displayCallback,
  channel_group: "sports",
  channel: "football,baseball,basketball,lacrosse,cricket"
});
```

Ahora, suscríbase al grupo de canales y se suscribirá a todos los canales en ese grupo:

```
pubnub.subscribe({
  callback: displayCallback,
  error: displayCallback,
  channel_group: "sports"
});
```

Todos los mensajes publicados en los canales del grupo de canales se recibirán en la función `displayCallback` .

Lea Controlador de flujo: Grupos de canales en línea: https://riptutorial.com/es/pubnub/topic/580/controlador-de-flujo--grupos-de-canales

Capítulo 4: Devolución de llamada específica de canal para v4 SDK

Examples

Devoluciones de llamada específicas del canal SDK de JavaScript v4

En PubNub JavaScript v3 , podría implementar una devolución de llamada única para cada canal al que se suscribió siempre y cuando haya llamado a la función de suscripción para cada canal e implementado la devolución de llamada en esa suscripción como esta:

```
var pubnub = new PubNub({
    publishKey: "your-pub-key",
    subscribeKey: "your-sub-key"
});

pubnub.subscribe({
    channel: 'ch1',
    message: function (m) {
        console.log(m + " ch1 callback");
    }
});

pubnub.subscribe({
    channel: 'ch2',
    message: function (m) {
        console.log(m + " ch2 callback: ");
    }
});
```

Entonces, en el código anterior, si publicas el mensaje `"hello"` en `ch1` :

```
publish({channel: "ch1", message: "hello"});
```

... entonces obtendrías la salida, `hello ch1 callback` . Y, por supuesto, si publicara el mismo mensaje en `ch2` , obtendría la `hello ch2 callback` .

La capacidad de proporcionar una devolución de llamada personalizado para cada canal, o un grupo de canales, es útil y, a menudo más deseable que la creación de una devolución de llamada monolítica con una larga `if-then-else` de nombres de canal y el código a ser ejecutado para cada condición. Y hay mejores prácticas para usar que el ejemplo simple anterior, pero quería facilitar la comparación y el contraste con el diseño de PubNub JavaScript SDK v4 (y cualquier v4 PubNub SDK).

Los SDK de PubNub v4 utilizan un único escucha para recibir todos los eventos de `message` , `presence` y `status` . Esto significa que cuando se `subscribe` a un canal, solo proporciona el (los) nombre (s) del canal en lugar de una función de `message callback` complementario. En JavaScript SDK v4 , el oyente se ve así:

```
pubnub.addListener({
    message: function(m) {
        console.log(JSON.stringify(m));
    },
    presence: function(p) {
        console.log(JSON.stringify(p));
    },
    status: function(s) {
        console.log(JSON.stringify(s));
    }
});
```

Sé que muchos desarrolladores se preguntarán cómo migrar el código de `subscribe` con `message callbacks` de `message callbacks` únicos desde SDK v3 con este tipo de diseño sin tener que recurrir al código condicional de nombre de canal que nunca finalicé que mencioné anteriormente, como este:

```
pubnub.addListener({
    message: function(m) {
        if (m.subscribedChannel == 'ch1') {
            console.log(m.message + "ch1 callback");
        }
        else if (m.subscribedChannel == 'ch2') {
            console.log(m.message + "ch2 callback");
        }
        else {
            console.log(m.message + "default callback");
        }
    }
    // removed the other two callbacks for brevity purposes
});
```

El parámetro `m` que se pasa a la escucha de mensajes anterior tiene la siguiente estructura, que es un diseño diferente al diseño de parámetros múltiples de JavaScript SDK v3.

```
{
    "channel": "ch1",
    "subscription": <undefined>,
    "timetoken": "14721821326909151",
    "message": "hello"
}
```

Ese `timetoken` es el `publish timetoken` real. Para los desarrolladores experimentados de PubNub, debería estar emocionado de ver que esto ahora está disponible para el suscriptor, pero no vamos a explicar por qué esto es importante y poderoso en este momento.

Ahora no esperaría que ningún desarrollador de JavaScript con experiencia escribiera el código como se muestra arriba y muchos desarrolladores avanzados ya sabrían qué hacer. Pero para aquellos desarrolladores que están en el nivel de principiante a intermedio con JavaScript, la solución puede no ser inmediatamente obvia. Sin embargo, sé que una vez que vea este sencillo enfoque de diseño a continuación, le abrirá los ojos a las posibilidades ilimitadas del lenguaje JavaScript: aquí vamos.

Replanteamos el requisito:

> Para cada canal al que me suscribo, quiero poder proporcionar una función única para invocar cuando se recibe un mensaje en ese canal. Y quiero evitar el enfoque de nombre de canal condicional monolítico.

Entonces, lo primero que debemos hacer es crear una tabla de búsqueda de funciones. Un `hashtable` para ser exactos. Esta tabla tendrá nombres de canales como teclas y funciones (el código que se invoca cuando se recibe un mensaje en ese canal) como valores. Si eres algo nuevo en JavaScript o has estado codificando con el idioma por un tiempo pero aún no te has metido en las funciones del idioma, esto puede sonar extraño e imposible, pero es realmente cómo funciona JavaScript y lo has estado haciendo todo el tiempo. y realmente no lo sabía. Vamos a definir nuestra tabla de búsqueda de funciones:

```
ftbl = {};
```

Eso es todo: tienes un objeto que mantendrá tus canales y funciones. Bastante simple, ¿verdad? Pero, ¿cómo agregar los canales y funciones? Al igual que cualquier otra clave / valor.

```
ftbl.ch1 = function(m){console.log(m.message + " ch1 callback")};
ftbl.ch2 = function(m){console.log(m.message + " ch1 callback")};
```

... y así sucesivamente con cada canal y función que desee definir. Y no tiene que crear todas sus teclas de función / canal en un lugar de su código. Puede agregar cada canal / función al `ftbl` al suscribirse a un canal.

```
ftbl.ch10 = function(m){console.log(m.message + " ch10 callback")};
pubnub.pubnub.subscribe({
    channels: ['ch10']
});
```

Ok, eso es lo suficientemente simple, y puede ser más sofisticado y avanzado con la forma en que lo hace, pero simplemente manteniéndolo básico. Pero, ¿cómo invoca esta función para el canal al que está vinculado? Esta es la razón por la que JavaScript es tan genial, potente y fácil, especialmente si proviene de un lenguaje rígido y estructurado como Java: échele un vistazo.

```
pubnub.addListener({
    message: function(m) {
        // use the channel name to get the function
        // from ftbl and invoke it
        ftbl[m.subscribedChannel](m);
    },
    presence: function(p) {
        console.log(JSON.stringify(p));
    },
    status: function(statusEvent) {
        console.log(JSON.stringify(s));
    }
});
```

Eso es todo al respecto. Simplemente obtenga la función de `ftbl` usando el nombre del canal que se pasa a la función de devolución de llamada del `message` del oyente y agregue `(m)` al final de la

misma y aumente, ejecuta su función.

Si el canal es `ch10` , `ftbl[m.subscribedChannel](m)` simplemente invoca la `function(m){console.log(m.message + "ch10 callback")}` pasando el parámetro `m` que su función puede analizar y explotar a medida que necesita.

Llamando así a la siguiente función de `publish` :

```
pubnub.publish(
    {
        channel : "ch10",
        message : "hello"
    },
    function(status, response) {
        console.log(status, response);
    }
);
```

... resultará en que se muestre el siguiente mensaje: `hello ch10 callback` . Y el equivalente para publicar en otros canales que haya definido en su tabla de búsqueda de funciones. No te olvides de proporcionar un valor predeterminado para canales desconocidos.

Y no olvide las devoluciones de llamada de `presence` y `status` en el oyente. Esto podría ser solo dos tablas de búsqueda de funciones más o simplemente un `ftbl` un poco más complejo:

```
ftbl.message.ch1 = function(m){console.log(m.message + " ch1 message cb")};
ftbl.presence.ch1 = function(m){console.log(m.message + " ch1 presence cb")};
ftbl.status.ch1 = function(m){console.log(m.message + " ch1 status cb")};
```

o

```
ftbl.ch1.message = ...
ftbl.ch1.presence = ...
ftbl.ch1.status = ...
```

Me gusta el primero mejor que el segundo, pero depende realmente de ti. Y es probable que desee algún código de manejo de eventos de `status` genérico de todos modos, pero dependerá de sus requisitos específicos.

Y esto puede ser aún más complicado y robusto con funciones opcionales por canal que pueden invocarse dependiendo de algunos datos adicionales que incruste en la carga útil del mensaje.

Así que ahí van, devoluciones de llamada únicas para cada canal. No más excusas para no querer migrar de 3x a 4x. Pero si tiene algunas dudas sobre la migración, no dude en comunicarse con el servicio de asistencia de PubNub y le ayudaremos a avanzar. Y no olvide revisar la Guía de migración de PubNub JavaScript SDK v3 a v4 .

Lea Devolución de llamada específica de canal para v4 SDK en línea: https://riptutorial.com/es/pubnub/topic/6037/devolucion-de-llamada-especifica-de-canal-para-v4-sdk

Capítulo 5: Filtrado de mensajes

Observaciones

Stream Filter ofrece la posibilidad de filtrar mensajes en el servidor antes de que se envíen a un suscriptor, es una solicitud popular. Con la introducción de nuestros SDK v4.x, ahora tiene la capacidad de hacerlo utilizando *metadatos de* mensajes.

Examples

Evite recibir sus propios mensajes usando Objective-C

La configuración de un filtro se aplica a todos los canales a los que se suscribirá desde ese cliente en particular. Este filtro de cliente *excluye los* mensajes que tienen el UUID de este suscriptor establecido en el UUID del remitente:

```
NSString *expression = [NSString stringWithFormat:@"(uuid != '%@'",
                        self.client.currentConfiguration.uuid];
[self.client setFilterExpression:expression];
```

Al publicar mensajes, debe incluir el UUID del remitente si desea que el filtro de cliente del lado del suscriptor funcione:

```
[self.client publish:@"message" toChannel:@"group-chat"
        withMetadata:@{@"uuid": self.client.currentConfiguration.uuid}
          completion:^(PNPublishStatus *status) {

    // Check whether request successfully completed or not.
    if (!status.isError) {

        // Message successfully published to specified channel.
    }
    else {

        // Request processing failed. Handle message publish error.
        // Check 'category' property to find out possible issue
        // publish can be attempted again using: [status retry];
    }
}];
```

Ver también:

- PubNub Objective-C SDK Stream Filter Documentación

Lea Filtrado de mensajes en línea: https://riptutorial.com/es/pubnub/topic/593/filtrado-de-mensajes

Capítulo 6: Gestor de acceso

Examples

Grupo de canales comodín Gestionar subvención - Java SDK v4

Cuando se trata de agregar / eliminar canales a / de sus grupos de canales, debe tener el permiso de `manage` para esos grupos de canales. Pero nunca debe otorgar a los clientes permiso para `manage` los grupos de canales a los que se suscribirán. Si lo hicieran, podrían agregar cualquier canal que quisieran a su grupo de canales y tener acceso de lectura instantáneo a ese canal.

Por esta razón, su servidor debe ser la única entidad que tiene el permiso de `manage` . Pero su servidor necesitará tener el permiso de `manage` para cada grupo de canales para que pueda agregar / eliminar canales a / desde grupos de canales en nombre de todos los clientes.

Pero otorgar la `manage` a cada grupo de canales puede ser un poco tedioso. En su lugar, puede otorgar `manage` a todos los grupos de canales (existentes y por crear) en una concesión de *comodín* .

```
// init PubNub instance using PNConfiguration with the secret-key
PNConfiguration pnConfiguration = new PNConfiguration();
pnConfiguration.setSubscribeKey("my_subkey")
pnConfiguration.setPublishKey("my_pubkey");
// secret key allows server to `grant` permissions
pnConfiguration.setSecretKey("my_secretkey");
pnConfiguration.setSecure(true);
// set the the server's auth key
pnConfiguration.setAuthKey("server_authkey");
PubNub pubnub = new PubNub(pnConfiguration);

// grant read and manage using the channel group wildcard - ":"
// with forever ttl (0)
pubNub.grant()
    .channelGroups(Arrays.asList(":")) // colon (:) is channel group wildcard
    .manage(true) // add/remove channels to/from channel groups
    .read(true) // in case server needs to subscribe or do here-now on channel groups
    .ttl(0) // 0 = forever grant
    .async(new PNCallback<PNAccessManagerGrantResult>() {
        @Override
        public void onResponse(PNAccessManagerGrantResult result, PNStatus status) {
            // check status for success or failure of grant
        }
    });
```

De aquí en adelante, su servidor podrá agregar / eliminar canales a / desde cualquier grupo de canales que cree su aplicación.

administrador de acceso de pubnub en el lado del servidor

PubNub Access Manager (PAM) amplía el marco de seguridad existente de PubNub al permitir a los desarrolladores crear y hacer cumplir el acceso seguro a los canales en toda la red en tiempo

real de PubNub.

Access Manager le permite administrar permisos granulares para sus aplicaciones y flujos de datos en tiempo real, crear múltiples niveles de permisos, otorgar y revocar el acceso y auditar el acceso de los usuarios.

Para usar Access Manager, debe habilitar Access Manager en el Panel de administración. Una vez que habilite Access Manager, debe otorgar permisos antes de poder enviar o recibir datos.

```
PAM Server side Configuration
```

Para que el lado del cliente funcione correctamente, el lado del servidor debe emitir primero los permisos adecuados para un canal PAM o una combinación de token de autenticación y grupo de canales.

para otorgar estos permisos, debe inicializar la instancia de pubnub al menos con sus suscripciones y claves secretas.

Ejemplo:

Paso 1. Hacer la configuración de Pubnub: -

```
PNConfiguration pnConfiguration = new PNConfiguration();
        pnConfiguration.setSubscribeKey(SUBSCRIBE_KEY);
        pnConfiguration.setPublishKey(PUBLISH_KEY);;
        pnConfiguration.setSecretKey(SECRET_KEY);
        pnConfiguration.setSecure(true);
        pnConfiguration.setLogVerbosity(PNLogVerbosity.BODY);
```

Paso 2. Inicializa PubNub con pnConfiguration

```
PubNub pubnub = new PubNub(pnConfiguration);
```

Operación PAM se produce a tres niveles

```
1. A global level (no auth key, and no channel/channel group is defined)

2. A channel/channel group level (only a channel/channel group is defined)

3. A channel/channel group and key level (where both the channel/channel     group and key are
defined)
```

En todos estos niveles podemos otorgar, revocar y auditar permisos. Aquí hacemos lo mismo en el canal / grupo de canales y nivel de clave de autenticación.

```
PAM  Grant
```

Podemos otorgar un permiso de lectura / escritura a auth_key en canales o grupos de canales específicos

Ejemplo:

Sincrónicamente:

```
try {

    pubnub.grant().authKeys(Arrays.asList("auth1,auth2"))
    .channels(Arrays.asList("channel1,channel2")).read(true).write(true    ).ttl(0).sync();

} catch (PubNubException e) {
    e.printStackTrace();
}
```

Asincrónicamente:

pubNub.grant (). canales (canales) .authKeys (Arrays.asList (authKey)). read (true) .write (true) .manage (false) .ttl (0) .async (new PNCallback () {

```
@Override
public void onResponse(PNAccessManagerGrantResult result,
     PNStatus status) {
}});
```

`PAM REVOKE:` podemos revocar un permiso para auth_key desde un canal o grupos de canales específicos.

Sintaxis para revocar el permiso igual que la concesión. Solo necesitamos cambiar el permiso de verdadero a falso.

```
try {

    pubnub.grant().authKeys(Arrays.asList("auth1,auth2"))
    .channels(Arrays.asList("channel1,channel2")).read(false).write( false    ).ttl(0).sync();

} catch (PubNubException e) {
    e.printStackTrace();
}
```

`PAM Audit:` podemos auditar un permiso dado para un canal / grupo de canales específico o para una determinada auth_key en un canal o grupo de canales específico

Ejemplo:

```
pubnub.audit().channel("mycha").authKeys(Arrays.asList("a1")).async(new
PNCallback<PNAccessManagerAuditResult>(){

    @Override
    public void onResponse(PNAccessManagerAuditResult result,
            PNStatus status) {

            }
    });
```

`PAM Add Channels into groups:` también podemos agregar canales en grupos de canales

Ejemplo:

```
pubnub.addChannelsToChannelGroup().channelGroup("my_channel").channels(Arrays.asList("my_channel5"))

    .async(new PNCallback<PNChannelGroupsAddChannelResult>() {

            @Override
            public void onResponse(PNChannelGroupsAddChannelResult
result,PNStatus status) {


            }
});
```

`Authentication Isue at Client Side (403 Forbidden):`

Si hay un error al realizar operaciones PAM, puede recibir un error 403. Si lo hace, asegúrese de haber configurado la clave secr_ correcta y que el reloj de la computadora emisora esté sincronizado con NTP.

```
NTP  Setup
```

El Protocolo de tiempo de red (NTP) es un protocolo que se utiliza para sincronizar los tiempos de reloj de la computadora en una red de computadoras. NTP usa el Tiempo Universal Coordinado (UTC) para sincronizar los tiempos de reloj de la computadora a un milisegundo y, a veces, a una fracción de un milisegundo.

Aquí necesitamos scyn server time con pubnub. Sigue el paso para hacerlo.

Paso 1 Instalación NTP

```
$ sudo apt-get update
$ sudo apt-get install ntp
```

Paso 2 Editar ntp.conf

```
Replace these four with pubnub server

server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org
```

a

```
server 0.pubsub.pubnub.com
server 1.pubsub.pubnub.com
server 2.pubsub.pubnub.com
server 3.pubsub.pubnub.com
```

Paso 3 Reinicie el servicio NTP

$ sudo service ntp restart

Ref:

[https://www.pubnub.com/docs/web-javascript/pam-security◆ [ ]]

https://www.pubnub.com/docs/java/pubnub-java-sdk-v4

Lea Gestor de acceso en línea: https://riptutorial.com/es/pubnub/topic/330/gestor-de-acceso

Capítulo 7: Hola Mundo

Observaciones

`<init>` (es decir, `require ("pubnub")`)

Inicializa una instancia de PubNub para invocar operaciones.

Parámetro	Detalles
publish_key	Cadena: su clave de publicación de su cuenta de PubNub Admin Dashboard
subscribe_key	Cadena: su clave de publicación de su cuenta de PubNub Admin Dashboard

`subscribe`

Suscríbase a un canal (es) y proporcione un medio para recibir mensajes publicados en el (los) canal (es).

Parámetro	Detalles
canal	Cadena: nombre del canal o lista delimitada por comas de los nombres de los canales
mensaje	Función: la función de devolución de llamada que recibirá mensajes publicados en los canales de suscripción.
conectar	Función: la función de devolución de llamada que se llamará cuando la suscripción a los canales sea exitosa.

`publish`

Publique un mensaje en un canal que será recibido por los suscriptores en ese canal.

Parámetro	Detalles
canal	Cadena - nombre del canal al que enviar el mensaje
mensaje	Cadena - El mensaje para publicar en el canal. Se recomienda el formato JSON (no aplicar una cadena de caracteres; use el objeto JSON)

Examples

Publicar y suscribirse para Node.JS SDK

Publicar y suscribirse para Node.JS

Instalar el paquete PubNub NPM.

```
npm install pubnub@3.15.2
```

Ejemplo de publicación Suscribir con Node.JS

```
var channel = "hello_world";
var pubnub = require("pubnub")({
    publish_key   : "your_pub_key"
,   subscribe_key : "your_sub_key"
});

pubnub.subscribe({
    channel : channel
,   message : receive  // print message
,   connect : send     // send message after subscribe connected
});

function receive(message) { console.log(message) }

function send(message) {
    pubnub.publish({
        channel : channel
    ,   message : message
    });
}
```

Lea Hola Mundo en línea: https://riptutorial.com/es/pubnub/topic/329/hola-mundo

Capítulo 8: Presencia

Examples

Presencia en JavaScript

La presencia funciona al enviar mensajes cuando un usuario se une, se retira o sale de un canal en particular. Puede escuchar estos mensajes para saber quién está en un canal y cuánto tiempo hace desde que hicieron algo.

En primer lugar, asegúrese de que cada usuario como UUID. Establece esto cuando inicias PubNub:

```
var pubnub = PUBNUB({
     publish_key: 'my_pub_key',
     subscribe_key: 'my_sub_key',
     uuid: '1234_some_uuid'
 });
```

Ahora, cuando se conecte a un canal, agregue una escucha adicional para eventos de `join` .

```
pubnub.subscribe({
    channel: "channel-1",
    message: function(m){console.log(m)}
    presence: onPresenceEvent,
});

onPresenceEvent = function(message, envelope, channel){
    if (!message.action) {
        // presence interval mode happens
        // when occupancy > presence announce max
        // there is no action key
        console.log("Presence Interval Mode: occupancy = " + m.occupancy);
        return;
    }

    console.log(
        "Action:    " + message.action + "\n" +
        "UUID:      " + message.uuid + "\n" +
        "Channel:   " + JSON.stringify(channel) + "\n" +
        "Occupancy: " + message.occupancy + "\n" +
        "Timestamp: " + message.timestamp);

    else if (m.action == 'join') {
        // new subscriber to channel
        // add the user to your buddy list
    }
    else if (m.action == 'leave') {
        // subscriber explicitly unsubscribed channel
        // remove user from your buddy list
    }
    else if (m.action == 'timeout') {
        // subscriber implicitly unsubscribed channel (did not unsubscribe)
        // remove user from your buddy list
```

```
    }
    else if (m.action == 'state-change') {
        // subscriber changed state
        // update the attributes about the user in the buddy list
        // i.e. - is typing, online status, etc.
        console.log("State Data: " + JSON.stringify(message.data));
    }
};
```

El objeto de mensaje enviado a la devolución de llamada de `presence` incluirá la acción realizada (unirse, salir, tiempo de espera o cambio de estado) y el UUID del usuario que realizó la acción, así como la marca de tiempo y algunos otros metadatos.

Cuando se establece el `state-change` se envía un evento de `state-change` que incluirá el nuevo estado en la clave de `data` clave de `message` .

Nota : si tiene Access Manager habilitado, *debe* asegurarse de que sus subvenciones cubren tanto el canal regular como el canal de presencia. De lo contrario, cuando intente suscribirse al canal con una devolución de llamada de presencia, el SDK también lo suscribirá al canal de presencia que fallará si no ha solicitado las subvenciones. El nombre del canal de presencia es el nombre del canal regular con un sufijo "-pnpres"; lo que significa que un canal llamado "pubnub-sensor-array" tendrá un canal de presencia llamado "pubnub-sensor-array-pnpres". Consulte los ejemplos de Access Manager para obtener más información.

Estado de configuración al suscribirse

Cuando un usuario se suscribe a un canal, es posible que desee establecer el estado para ese usuario recién suscrito. Si bien hay una *suscripción a la* API del *estado* , hay algunos escenarios en los que esta no es la técnica más óptima / confiable (como durante una situación de desconexión / reconexión: el estado se perderá y no se restablecerá).

Es mejor establecer explícitamente el estado una vez que el canal se haya suscrito correctamente. Esto significa que utiliza la devolución de llamada de `connect` del `subscribe` para establecer el estado.

```
var pubnub = PUBNUB({
    publish_key: 'my_pub_key',
    subscribe_key: 'my_sub_key',
    uuid: 'users_uuid'
});

pubnub.subscribe({
    channel: 'channel-1',
    message: function(msg, env, ch){console.log(msg)},
    connect: function(m) {
        console.log('CONNECT: ' + m);
        pubnub.state({
            channel  : 'channel-1', // use the channel param from the subscribe
            state    : {'nickname': 'Bandit', 'mood': 'Pumped!'},
            callback : function(m){console.log(m)},
            error    : function(m){console.log(m)}
        });
    },
```

```
    disconnect : function(m){console.log('DISCONNECT: ' + m)},
    reconnect  : function(m){console.log('RECONNECT: ' + m)},
    error      : function(m){console.log('CONNECT: ' + m)}
});
```

Lea Presencia en línea: https://riptutorial.com/es/pubnub/topic/331/presencia

Capítulo 9: UUIDs

Examples

JavaScript / Web SDK

Para JavaScript, aquí está el código que recomendamos para generar, persistir y recuperar un UUID. Esto podría envolverse en una función que se puede llamar directamente desde la función PUBNUB.init en lugar de la solución en línea de dos pasos a continuación.

```
// get/create/store UUID
var UUID = PUBNUB.db.get('session') || (function(){
    var uuid = PUBNUB.uuid();
    PUBNUB.db.set('session', uuid);
    return uuid;
})();

// init PUBNUB object with UUID value
var pubnub = PUBNUB.init({
    publish_key: pubKey,
    subscribe_key: subKey,
    uuid: UUID
});
```

Android / Java SDK

Para Android, aquí está el código que recomendamos para generar, persistir y recuperar un UUID. No hay un constructor que acepte el UUID como parámetro, por lo que primero debe crear `Pubnub` instancia del objeto `Pubnub` luego usar el configurador para proporcionar el UUID.

```
// creating the Pubnub connection object with minimal args
Pubnub pubnub = new Pubnub(pubKey, subKey);

// get the SharedPreferences object using private mode
// so that this uuid is only used/updated by this app
SharedPreferences sharedPrefs = getActivity().getPreferences(Context.MODE_PRIVATE);

// get the current pn_uuid value (first time, it will be null)
String uuid = getResources().getString(R.string.pn_uuid);

// if uuid hasn't been created & persisted, then create
// and persist to use for subsequent app loads/connections
if (uuid == null || uuid.length == 0) {
    // PubNub provides a uuid generator method but you could
    // use your own custom uuid, if required
    uuid = pubnub.uuid();
    SharedPreferences.Editor editor = sharedPrefs.edit();
    editor.putString(getString(R.string.pn_uuid), uuid);
    editor.commit();
}

// set the uuid for the pubnub object
pubnub.setUUID(uuid);
```

Lea UUIDs en línea: https://riptutorial.com/es/pubnub/topic/561/uuids

Capítulo 10: webhook

Examples

pubhub webhook

```
Webhook Overview
```

Un WebHook es una devolución de llamada HTTP: un HTTP POST que ocurre cuando ocurre algo; una simple notificación de eventos a través de HTTP POST. Una aplicación web que implemente WebHooks enviará un mensaje a una URL cuando ocurran ciertas cosas.

```
PubNub Presence
```

Presencia de pubnub todo sobre la presencia del usuario en la plataforma de pubnub. proporciona la presencia del usuario cuando se está uniendo, dejando un canal o cuando hay cambios de estado de un usuario. Presence Webhooks proporciona un medio para que su servidor sea notificado cuando ocurran eventos de presencia en cualquier canal para sus claves. Esto proporciona una solución fácil de escalar para su aplicación del lado del servidor para monitorear los eventos de presencia.

```
How it would reduce the overhead
```

Sin Presence Webhooks, su servidor tendría que suscribirse a todos los canales -pnpres canales. Por lo tanto, esta puede ser una tarea tediosa para controlar canales externos si su aplicación tiene miles de canales o más.

Pubnub Webhooks nos ayudaría en este escenario y es más fácil de implementar y escalar con una infraestructura web tradicional bien conocida (balanceadores de carga, servidores web y de aplicaciones proporcionados por sus proveedores de servicios de aplicaciones como Heroku, Rackspace, Azure, Amazon y otros).

```
PubNub Presence Webhooks
```

Los WebHooks de presencia de PubNub son un medio para que la red de PubNub invoque un punto final REST de su servidor directamente a medida que ocurren los eventos de presencia. También ayudaría en el equilibrio de carga. Entonces, aquí necesita crear la URL de punto final de REST de su servidor a la que pubnub enviaría los datos de presencia.

```
User Presence Events
```

Hay cuatro eventos de usuario en la plataforma pubnunb.

1. unirse
2. salir

3. se acabó el tiempo
4. cambio de estado

Y dos eventos de nivel de canal: activo e inactivo. Consulte pubnub-doc para obtener información detallada.

Cada evento tiene su propio Webhook para el cual puede proporcionar un punto final REST a su servidor para manejar el evento. O puede proporcionar un punto final REST para todos ellos y simplemente implementar la lógica condicional en el atributo de acción en su servidor para manejar cada evento individual.

Independientemente de lo que elija, debe proporcionar la subclave y los URI de REST al Soporte de PubNub para configurarlo por usted. Es probable que tenga más de una subclave con diferentes puntos finales para diferentes entornos de servidor (desarrollo, prueba, producción, por ejemplo).

Una vez que se implementan los puntos finales REST de su servidor y la configuración de la clave PubNub está en su lugar, está listo para comenzar. Pero antes de implementar los puntos finales REST, puede ser útil saber cómo se ven los datos de los eventos.

Aquí hay un ejemplo de una unión:

```
HTTP POST
Content-Type: application/json

{
    'action': 'join',
    'sub_key': 'sub-c-...',
    'channel': 'lacrosse'
    'uuid': '1234-5678-90ab-cdef',
    'timestamp': 1440568311,
    'occupancy': 1,
    'data': {'foo': 'bar'}
}
```

Esto sería lo mismo para la licencia, el tiempo de espera y el cambio de estado, excepto por el valor de la acción, por supuesto.

```
Webhook Response Status
```

Es importante que su implementación de punto final REST devuelva un código de estado (200 OK) inmediatamente después de recibir el Webhook de PubNub.

```
Pubnub Re-try
```

Si pubnub no recibe 200 From Rest-Endpoint, enviará eventos duplicados porque PubNub no asume ninguna respuesta significa que su servidor no recibió el evento. PubNub esperará cinco segundos la respuesta 200 antes de volver a intentarlo. Después de un tercer reintento (cuatro intentos en total), PubNub ya no intentará enviar ese evento en particular a su servidor.

Lea webhook en línea: https://riptutorial.com/es/pubnub/topic/3715/webhook

Creditos

S. No	Capítulos	Contributors
1	Empezando con PubNub	Community, Craig Conover, Dara Kong, Josh Marinacci
2	Almacenamiento y reproducción	Craig Conover
3	Controlador de flujo: Grupos de canales	Craig Conover
4	Devolución de llamada específica de canal para v4 SDK	Craig Conover
5	Filtrado de mensajes	Craig Conover, Serhii Mamontov
6	Gestor de acceso	Craig Conover, Girish Kumar, Josh Marinacci
7	Hola Mundo	Craig Conover, girlie_mac, PubNub
8	Presencia	Alex Vidal, Craig Conover, Dara Kong, Josh Marinacci
9	UUIDs	Craig Conover
10	webhook	Girish Kumar