example TypeScript script using PnP JS for a Solution Package Version History Checking and Alert Mechanism in SharePoint.

This script can be used in an SPFx extension or Node.js script with PnP JS to check the App Catalog for outdated solutions and trigger alerts.

TypeScript Code: Solution Package Version Checker

typescript

Copy code

import { sp } from "@pnp/sp/presets/all";

import "@pnp/sp/webs";

import "@pnp/sp/lists";

import "@pnp/sp/items";

import "@pnp/sp/sites";

import "@pnp/sp/appcatalog";

// Define interface for solution package

interface SolutionPackage {

Title: string; // Solution Name

SolutionVersion: string; // Current Version

Status: string; // Active / Deprecated

SiteUrl: string; // Site where deployed

LastUpdated: string; // Last update timestamp

}

// Function to compare versions

function isVersionOutdated(current: string, latest: string): boolean {

const currentParts = current.split(".").map(Number);

const latestParts = latest.split(".").map(Number);

for (let i = 0; i < latestParts.length; i++) {

if ((currentParts[i] || 0) < latestParts[i]) {

return true;

} else if ((currentParts[i] || 0) > latestParts[i]) {

return false;

}

}

return false;

}

// Main function to check solution versions

export async function checkSolutionVersions() {

try {

// Connect to SharePoint

console.log("Checking solution package versions...");

// 1. Get all solutions from App Catalog

const appCatalogSolutions = await sp.site.getAppCatalog().availableApps();

console.log("App Catalog Solutions Found:", appCatalogSolutions.length);

for (let app of appCatalogSolutions) {

const solutionName = app.Title;

const latestVersion = app.InstalledVersion;

console.log(\nSolution: ${solutionName} | Latest Version: ${latestVersion});

// 2. Query all site collections for this solution

const sites = await sp.site.getAllSites(); // Requires Admin Permissions

for (let site of sites) {

const siteUrl = site.Url;

// Get installed apps in the site

const siteApps = await sp.site.getAppCatalog().installedApps();

const siteApp = siteApps.find((sa: any) => sa.Title === solutionName);

if (siteApp) {

const currentVersion = siteApp.InstalledVersion;

// Compare versions

if (isVersionOutdated(currentVersion, latestVersion)) {

console.warn(⚠ Outdated version found at ${siteUrl}: ${currentVersion});

// Optional: Trigger alert to admins

await sendAlertToAdmins(solutionName, currentVersion, latestVersion, siteUrl);

}

}

}

}

console.log("Version check completed.");

} catch (error) {

console.error("Error checking solution versions:", error);

}

}

// Function to send alert (placeholder)

async function sendAlertToAdmins(

solutionName: string,

currentVersion: string,

latestVersion: string,

siteUrl: string

) {

// Example: Add an item to a SharePoint Alerts List

await sp.web.lists.getByTitle("SolutionVersionAlerts").items.add({

Title: solutionName,

CurrentVersion: currentVersion,

LatestVersion: latestVersion,

SiteUrl: siteUrl,

AlertDate: new Date().toISOString(),

Status: "Outdated"

});

console.log(Alert added for ${solutionName} in site ${siteUrl});

}

// Execute script

checkSolutionVersions();

How It Works

Connects to the SharePoint App Catalog using PnP JS.

Fetches all solution packages and their latest versions.

Loops through all site collections to check installed solution versions.

Compares current version with the latest App Catalog version.

Adds an alert to a SharePoint list if outdated solutions are found.

Prerequisites

SPFx or Node.js environment with PnP JS installed:

bash

Copy code

npm install @pnp/sp @pnp/nodejs --save

Proper Admin Permissions to read all site collections.

A SharePoint list (e.g., SolutionVersionAlerts) to store alerts.

**Problem Statement: Townhall Scheduling and Broadcast in SPFx**

Organizations conduct Townhall meetings regularly for employee communication.

Currently, details are shared manually via emails, which leads to low visibility and last-minute confusion.

Goal:

Schedule Townhall events in SharePoint.

Automatically publish a broadcast thread in Microsoft Teams when the event is scheduled.

Maintain a SharePoint list of all scheduled Townhalls with event details and broadcast status.

Requirements

SharePoint List – TownhallSchedule

Columns: Title, Date, Time, Description, TeamsThreadLink, BroadcastStatus

SPFx Script / Web Part to:

Schedule a new Townhall (add to list).

Publish a broadcast thread to a designated Teams channel using Graph API.

Notifications to all team members via Teams.

TypeScript Code: Townhall Scheduling & Broadcast

typescript

Copy code

import { sp } from "@pnp/sp/presets/all";

import "@pnp/sp/webs";

import "@pnp/sp/lists";

import "@pnp/sp/items";

import { graph } from "@pnp/graph/presets/all";

import "@pnp/graph/teams";

// Define Townhall Event

interface TownhallEvent {

Title: string;

Date: string;

Time: string;

Description: string;

TeamsThreadLink?: string;

BroadcastStatus?: string;

}

// Configure Teams Channel

const TEAM\_ID = "<Your-Team-ID>";

const CHANNEL\_ID = "<Your-Channel-ID>";

// Function to schedule a townhall

export async function scheduleTownhall(event: TownhallEvent) {

try {

console.log("Scheduling Townhall:", event.Title);

// 1. Add to SharePoint list

const item = await sp.web.lists.getByTitle("TownhallSchedule").items.add({

Title: event.Title,

EventDate: event.Date,

EventTime: event.Time,

Description: event.Description,

BroadcastStatus: "Scheduled"

});

const itemId = item.data.Id;

console.log(Townhall added to SharePoint with ID: ${itemId});

// 2. Publish broadcast thread in Teams

const broadcastThread = await publishBroadcastToTeams(event);

if (broadcastThread && broadcastThread.id) {

const threadLink = https://teams.microsoft.com/l/message/${TEAM\_ID}/${CHANNEL\_ID}/${broadcastThread.id};

// 3. Update SharePoint list with broadcast info

await sp.web.lists.getByTitle("TownhallSchedule").items.getById(itemId).update({

TeamsThreadLink: threadLink,

BroadcastStatus: "Broadcasted"

});

console.log(Broadcast thread created: ${threadLink});

}

} catch (error) {

console.error("Error scheduling Townhall:", error);

}

}

// Function to publish broadcast thread to Teams channel

async function publishBroadcastToTeams(event: TownhallEvent) {

try {

const message = {

body: {

content: `<b>📢 Upcoming Townhall:</b> ${event.Title}<br>

🗓 Date: ${event.Date}<br>

⏰ Time: ${event.Time}<br>

📝 ${event.Description}<br>

Please join and participate!`

}

};

const response = await graph.teams.getById(TEAM\_ID)

.channels.getById(CHANNEL\_ID)

.messages.add(message);

console.log("Teams broadcast posted successfully.");

return response;

} catch (error) {

console.error("Error broadcasting to Teams:", error);

return null;

}

}

// Example Usage

scheduleTownhall({

Title: "Q3 Townhall",

Date: "2025-08-15",

Time: "4:00 PM",

Description: "Join us for company updates and Q&A with leadership."

});

How It Works

User schedules a Townhall → Adds event details to TownhallSchedule list.

Script creates a Teams post → A broadcast thread in the designated Teams channel.

SharePoint list is updated → Thread link and broadcast status are logged.

Employees see the broadcast → Clickable Teams thread for easy access.

Prerequisites

SPFx solution with PnP JS + Graph API configured.

Teams app registration with ChannelMessage.Send permission.

SharePoint List TownhallSchedule with columns:

Title, EventDate, EventTime, Description, TeamsThreadLink, BroadcastStatus

Extensions Possible

Automated Reminder – Use Power Automate

Requirements

SharePoint List – TownhallSchedule

Columns: Title, Date, Time, Description, TeamsThreadLink, BroadcastStatus

SPFx Script / Web Part to:

Schedule a new Townhall (add to list).

Publish a broadcast thread to a designated Teams channel using Graph API.

Notifications to all team members via Teams.

TypeScript Code: Townhall Scheduling & Broadcast

typescript

Copy code

import { sp } from "@pnp/sp/presets/all";

import "@pnp/sp/webs";

import "@pnp/sp/lists";

import "@pnp/sp/items";

import { graph } from "@pnp/graph/presets/all";

import "@pnp/graph/teams";

// Define Townhall Event

interface TownhallEvent {

Title: string;

Date: string;

Time: string;

Description: string;

TeamsThreadLink?: string;

BroadcastStatus?: string;

}

// Configure Teams Channel

const TEAM\_ID = "<Your-Team-ID>";

const CHANNEL\_ID = "<Your-Channel-ID>";

// Function to schedule a townhall

export async function scheduleTownhall(event: TownhallEvent) {

try {

console.log("Scheduling Townhall:", event.Title);

// 1. Add to SharePoint list

const item = await sp.web.lists.getByTitle("TownhallSchedule").items.add({

Title: event.Title,

EventDate: event.Date,

EventTime: event.Time,

Description: event.Description,

BroadcastStatus: "Scheduled"

});

const itemId = item.data.Id;

console.log(Townhall added to SharePoint with ID: ${itemId});

// 2. Publish broadcast thread in Teams

const broadcastThread = await publishBroadcastToTeams(event);

if (broadcastThread && broadcastThread.id) {

const threadLink = https://teams.microsoft.com/l/message/${TEAM\_ID}/${CHANNEL\_ID}/${broadcastThread.id};

// 3. Update SharePoint list with broadcast info

await sp.web.lists.getByTitle("TownhallSchedule").items.getById(itemId).update({

TeamsThreadLink: threadLink,

BroadcastStatus: "Broadcasted"

});

console.log(Broadcast thread created: ${threadLink});

}

} catch (error) {

console.error("Error scheduling Townhall:", error);

}

}

// Function to publish broadcast thread to Teams channel

async function publishBroadcastToTeams(event: TownhallEvent) {

try {

const message = {

body: {

content: `<b>📢 Upcoming Townhall:</b> ${event.Title}<br>

🗓 Date: ${event.Date}<br>

⏰ Time: ${event.Time}<br>

📝 ${event.Description}<br>

Please join and participate!`

}

};

const response = await graph.teams.getById(TEAM\_ID)

.channels.getById(CHANNEL\_ID)

.messages.add(message);

console.log("Teams broadcast posted successfully.");

return response;

} catch (error) {

console.error("Error broadcasting to Teams:", error);

return null;

}

}

// Example Usage

scheduleTownhall({

Title: "Q3 Townhall",

Date: "2025-08-15",

Time: "4:00 PM",

Description: "Join us for company updates and Q&A with leadership."

});

How It Works

User schedules a Townhall → Adds event details to TownhallSchedule list.

Script creates a Teams post → A broadcast thread in the designated Teams channel.

SharePoint list is updated → Thread link and broadcast status are logged.

Employees see the broadcast → Clickable Teams thread for easy access.

Prerequisites

SPFx solution with PnP JS + Graph API configured.

Teams app registration with ChannelMessage.Send permission.

SharePoint List TownhallSchedule with columns:

Title, EventDate, EventTime, Description, TeamsThreadLink, BroadcastStatus

Extensions Possible

Automated Reminder – Use Power Automate or Azure Function to post reminders 1 hour before the event.

Adaptive Card Broadcast – Post a rich Adaptive Card in Teams instead of plain text.

Calendar Integration – Sync event to Outlook calendar automatically.