GitHub CodeSpaces

Building a PCF Client Bandwidth Checker Using GitHub Codespaces: An End-to-End Guide

Author: Rohan Wadiwala

Contents

Introduction

As digital transformation accelerates, application responsiveness and performance are more crucial than ever. In Power Platform, custom Power Apps Component Framework (PCF) controls empower makers to add rich, device-specialized logic to their apps. What if you need to detect your user's bandwidth in real time? In this guide, you'll learn how to build, test, and deploy a custom PCF control for bandwidth detection—completely in the cloud—using GitHub Codespaces.

The Case for Developing PCF Controls in GitHub Codespaces

As the Power Platform ecosystem expands, development teams face increasing challenges in managing complex environments, dependencies, and collaborative workflows—particularly when building custom Power Apps Component Framework (PCF) controls. Utilizing GitHub Codespaces as your primary development environment is not merely a matter of convenience; it represents a modern, cloud-first approach that offers significant advantages over traditional local development.

Key Benefits and Rationale

- **Consistent, Pre-Configured Environments**
  GitHub Codespaces provides instantly provisioned, containerized development environments that are consistent across all users. This eliminates the pervasive "works on my machine" issues, ensuring every developer operates in an identical workspace and reducing configuration discrepancies.

- **Elimination of Local Setup Burdens**
  Developers are no longer required to manually install or manage dependencies such as specific versions of Node.js, the Power Platform CLI, or the .NET SDK. Codespaces provisions these tools automatically, substantially reducing onboarding time and mitigating environment-related errors.

- **True Cross-Platform Accessibility**
  Whether a team member uses Windows, macOS, Linux, or even a Chromebook, Codespaces guarantees the same development experience through any modern browser. This universality democratizes access to Power Platform development and minimizes OS-specific dependency challenges.

- **Streamlined Collaboration and Onboarding**
  Facilitating team collaboration is greatly simplified: new contributors or colleagues can readily launch a standardized development environment via your repository. This accelerates onboarding, troubleshooting, and code reviews, as all users share an inherently aligned workspace.

- **Integrated Source Control**
  Development within Codespaces seamlessly integrates with GitHub repositories. Actions such as committing, branching, or collaborating occur naturally within the environment, removing the need for additional configuration or external tools.

- **Enhanced Security and Isolation**
  Running your development environment in the cloud ensures that potentially risky code and external package installations occur outside your personal device, thereby reducing exposure to vulnerabilities and safeguarding your local assets.

- **On-Demand Resource Scalability**
  Codespaces allows for flexible adjustment of computational resources according to project demands. This is particularly beneficial for building or packaging resource-intensive PCF solutions, enabling efficient utilization without overburdening local hardware.

- **Mobility and Operational Flexibility**
  Should you need to transition between devices or work from an alternate location, Codespaces enables you to pause and resume environments seamlessly, preserving your workflow regardless of hardware or geographic constraints.

- **Automatic Maintenance and Updates**
  Microsoft manages all security updates, OS patches, and base image maintenance for Codespaces. Developers are therefore insulated from unexpected disruptions due to environment drift or missing updates.

Comparison with Traditional Local Development

Aspect	GitHub Codespaces	Local Development
Environment Consistency	Guaranteed across all users	Variable and prone to drift
Setup and Onboarding Time	Minutes	Hours or more per developer
Platform Accessibility	Fully cross-platform (browser-based)	Dependent on local OS
Collaboration	Effortless via shared repositories	Manual setup for each collaborator
Resource Management	Dynamically scalable in the cloud	Limited by local machine specs
Security	Isolated and sandboxed	Direct exposure of local system
Source Control Integration	Native and seamless	May require additional configuration
Environment Recovery	Instant re-provisioning	Manual and time-consuming

In conclusion, adopting GitHub Codespaces for PCF control development not only enhances efficiency but also supports organizational goals related to security, scalability, and collaboration. It streamlines the development process, enabling teams to focus on delivering high-quality customizations for the Power Platform while minimizing operational overhead. This makes Codespaces a recommended practice for modern, cloud-oriented Power Platform development initiatives.

Let’s dive into the full process, step by step.

1. Overview & Prerequisites

Before starting, make sure you have:

- **A valid GitHub account** with access to Codespaces (beta or general availability).
- **A Power Platform environment** (with admin or maker rights).
- Familiarity with TypeScript, Node.js, and PCF basics.
- Optionally, Visual Studio Code if you prefer editing locally.

For our purpose, everything will be cloud-based, harnessing Codespaces’ full capability.

2. Repository Setup in GitHub

Let’s start by creating a fresh repository to organize our code and assets:

1. **Navigate to GitHub** and log in.
2. Click the **"+"** icon in the upper right and select **"New repository"**.
3. Choose a name—such as pcf-bandwidth-control.
4. Add a description, such as:
   “PCF control to measure and display client-side bandwidth in Power Apps.”
5. Check **"Initialize this repository with a README"** (recommended).

6. Ensure, .gitignore is not configured.
7. Click **Create repository**.

This sets up your code hub and unlocks Codespaces for rapid cloud development.

3. Spinning Up GitHub Codespaces

Codespaces allows instant cloud dev environments. Here's how to get started:

1. Go to your new repo.
2. Click the green **"Code"** button, then the **"Codespaces"** tab.
3. Click **"Create codespace on main"** (or your desired branch).
4. Wait a minute while Codespaces provisions everything.

You'll soon see a VS Code-like cloud IDE—no installs, no hassle.

4. Configuring the Development Environment

Let's ensure we have all the required tools set up.

a. Verify Node.js and npm

Open the Codespaces terminal and run:

```
node -v
npm -v
```

If either isn't recognized, install with:

```
sudo apt update
sudo apt install nodejs npm
```

b. Install Power Platform CLI

Microsoft's CLI is key for scaffolding and packaging PCF controls.

```
npm install -g @microsoft/powerplatform-cli
```

In case you get error while installing CLI via. npm then try installing it via. dotnet

```
dotnet tool install --global Microsoft.PowerApps.CLI.Tool --version 1.36.3
```

(use this version for .NET 8)

Once done, check installation via:

```
pac --version
```

c. Install .NET SDK

Required for building and packing solutions. To add it:

```
sudo apt-get install apt-transport-https
wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt-get update
sudo apt-get install dotnet-sdk-6.0
dotnet --version
```

Tip: *Codespaces generally comes with Node and .NET preinstalled. These steps are for validation or recovery.*

5. Creating the PCF Bandwidth Control

Let's scaffold our custom PCF project.

1. In the terminal, navigate to your repo root.
2. Initialize:

```
pac pcf init --namespace BandwidthDemo --name BandwidthCheckControl --template field
```

- --namespace: Your organization's or app's logical space.
- --name: A unique name for your PCF control.
- --template: Most use **field** for data-bound controls.

3. This creates a subfolder with your control. Move inside:\

```
cd BandwidthCheckControl
```

4. Install dependencies:

```
npm install
```

You're ready to develop!

6. Implementing Bandwidth Measurement Logic

Now, we make the control actually check client bandwidth.

a. The Approach

A common method: download a known image and measure the time it takes. This gives a quick, reasonably accurate indication of available network speed.

b. Code Example

Open index.ts and replace the code to below:

```
export class BandwidthCheckControl implements ComponentFramework.StandardControl<IInputs,
IOutputs> {

  private _container: HTMLDivElement;
  private _bandwidthDisplay: HTMLSpanElement;
  private _notifyOutputChanged: () => void;
  private bandwidthKbps: number = 0;
```

```
  constructor() {
  }

  /**
   * Used to initialize the control instance
   * @param context Provides data and environment information
   * @param notifyOutputChanged A callback method to notify the framework that outputs have
changed
   * @param state A control state dictionary
   * @param container An HTML Div element where control UI can be added
   */
  public init(
    context: ComponentFramework.Context<IInputs>,
    notifyOutputChanged: () => void,
    state: ComponentFramework.Dictionary,
    container: HTMLDivElement
  ): void {
    this._container = document.createElement("div");
    this._container.style.fontFamily = "Segoe UI, Tahoma, Geneva, Verdana, sans-serif";

    const label = document.createElement("label");
    label.textContent = "Estimated Bandwidth (Kbps): ";
    label.style.fontWeight = "600";

    this._bandwidthDisplay = document.createElement("span");
    this._bandwidthDisplay.textContent = "Measuring...";

    this._container.appendChild(label);
    this._container.appendChild(this._bandwidthDisplay);

    container.appendChild(this._container);

    this._notifyOutputChanged = notifyOutputChanged;

    // Start measurement immediately
    this.measureAndUpdateBandwidth();
  }

  /**
   * Called when any value in the property bag has changed
   * @param context Provides updated data and environment info
   */
  public updateView(context: ComponentFramework.Context<IInputs>): void {
    // Optionally, you could re-trigger bandwidth measurement here based on context changes,
    // but for this example, we only measure on init.
  }

  /**
   * Called by the framework to get updated outputs
   */
  public getOutputs(): IOutputs {
    return {
      bandwidthKbps: this.bandwidthKbps
    };
  }

  /**
   * Called when the control is to be removed
   */
  public destroy(): void {
    // Clean up event listeners or timers here if any
  }
```

```
  /**
   * Measure client bandwidth by downloading an image and timing it
   */
  private checkBandwidth(): Promise<number> {
    return new Promise((resolve) => {
      const imageAddr =
"https://upload.wikimedia.org/wikipedia/commons/3/3f/Fronalpstock_big.jpg";
      const startTime = new Date().getTime();
      const download = new Image();

      download.onload = () => {
        const endTime = new Date().getTime();
        const durationSeconds = (endTime - startTime) / 1000;
        const bitsLoaded = 16000000; // approx 2MB image in bits
        const bps = bitsLoaded / durationSeconds;
        const kbps = bps / 1024;

        resolve(Math.round(kbps));
      };

      download.onerror = () => {
        resolve(0);
      };

      // Add cache buster to avoid cached images influencing measurement
      download.src = imageAddr + "?cacheBust=" + Math.random();
    });
  }

  /**
   * Helper method to run the bandwidth check and update UI and output
   */
  private async measureAndUpdateBandwidth(): Promise<void> {
    this._bandwidthDisplay.textContent = "Measuring...";
    try {
      const kbps = await this.checkBandwidth();
      this.bandwidthKbps = kbps;

      if (kbps === 0) {
        this._bandwidthDisplay.textContent = "Measurement failed";
      } else {
        this._bandwidthDisplay.textContent = `${kbps.toLocaleString()} Kbps`;
      }
    } catch {
      this._bandwidthDisplay.textContent = "Error measuring bandwidth";
      this.bandwidthKbps = 0;
    }

    // Notify framework outputs have changed
    this._notifyOutputChanged();
  }
}
```

Add the following line: ControlManifest.Input.xml

```
<control. . .>


<property name="bandwidthKbps" display-name-key="BandwidthKbps_DisplayKey" description-
key="BandwidthKbps_DescKey" of-type="Whole.None" usage="output" />
```

```
...
</control>
```

7. Building and Testing Locally

With the control logic in place, let's compile and run it in a safe, local harness.

a. Build the PCF

```
npm run build
```

b. Launch the PCF Test Harness

Still from the control folder:

```
npm start
```

This opens the PCF test harness in Codespaces' browser preview. Test your control by interacting with the simulated field. Check if bandwidth values update and handle error cases (like offline mode).

8. Packaging & Deploying to Power Platform

Once satisfied with local testing, package and deploy your PCF control.

a. Push Control Directly (For Dev/Test)

If you want to quickly see your control in Power Apps:

```
pac pcf push --publisher-prefix yourprefix
```

b. Package as a Solution (Production/Sharing)

1. Back in your terminal, from the repo root:

```
pac solution init --publisher-name "MyPublisher" --publisher-prefix "mypub"
```

2. Add the PCF project:

```
pac solution add-reference --path BandwidthCheckControl
```

You'll get a .zip in the output directory—this is the solution file ready to import.

9. Adding and Testing Your Control in Power Apps

1. Go to your Power Apps environment.
2. Navigate to **Solutions**, then **Import**, and upload your new .zip solution.
3. Once imported, it will show in custom controls.

4. Edit or create a model-driven app form. On a supported field, set your new PCF control as the visualization.

5. Save and **Publish**.

Try accessing your app from different devices or networks to see live bandwidth readings reflected directly in your business apps!

10. CI/CD and Maintenance

Automating Quality

- Use **GitHub Actions** for workflows like:
    - Install dependencies
    - Run npm run build
    - Push/publish/test solution
- Document dependencies and environments in a README.md.
- Version control your solution files for easy rollback and audit.

Keeping Up-to-Date

- Frequently update dependencies (npm update).
- Monitor changes in Power Platform CLI and PCF runtime APIs.

11. Conclusion

Through this cloud-first workflow, you've created a modern, dynamic PCF control that enables Power Apps to smartly react to real-world client network conditions.

Using GitHub Codespaces allows for fast, replicable, and secure development anywhere, without the overhead of local setup. By following this guide, you're ready to empower makers and users with context-aware apps that perform their best—everywhere, every time.

About Sogeti

Sogeti is a leading provider of technology and engineering services. Sogeti delivers solutions that enable digital transformation and offers cutting-edge expertise in Cloud, Cybersecurity, Digital Manufacturing, Digital Assurance & Testing, and emerging technologies. Sogeti combines agility and speed of implementation with strong technology supplier partnerships, world class methodologies and its global delivery model, Rightshore®. Sogeti brings together more than 25,000 professionals in 15 countries, based in over 100 locations in Europe, USA and India. Sogeti is a wholly-owned subsidiary of Capgemini SE, listed on the Paris Stock Exchange.

Learn more about us at

www.sogeti.com