



Understanding LaunchedEffect and rememberCoroutineScope in Jetpack Compose

Jetpack Compose is declarative — your **Composable functions can re-run (recompose)** whenever state changes.

When you need to run **side effects** (like network calls, reading from disk, or showing a toast), you must be careful not to accidentally trigger them on every recomposition.

Two common ways to launch coroutines in Compose are:

-  LaunchedEffect
-  rememberCoroutineScope

But they are used **for different purposes**.

What is LaunchedEffect

LaunchedEffect is a **side-effect API** that runs a coroutine **when a Composable enters the composition or when one of its keys changes**.

```
LaunchedEffect(key1, key2, ...) {  
    // Runs once when first composed, then again if any key changes  
}
```

- It automatically gives you a **CoroutineScope** tied to the Composable's lifecycle.
- Cancels the coroutine automatically when the Composable leaves the screen.
- Restarts the block when the **keys** you pass change.

Behavior

Code	Runs when
LaunchedEffect(Unit)	Once when the Composable first appears.
LaunchedEffect(someValue)	Once on first composition and whenever someValue changes.
LaunchedEffect(key1, key2)	Once on first composition and whenever any key changes .

 It **does NOT** run again on every normal recomposition if the keys don't change.

Example

```

@Composable
fun DemoLaunchedEffect() {
    var count by remember { mutableStateOf(0) }

    LaunchedEffect(Unit) {
        println("Runs only once")
    }

    Button(onClick = { count++ }) {
        Text("Count = $count")
    }
}

```

- Clicking the button changes `count` → causes recomposition.
- But `LaunchedEffect(Unit)` **does not restart** (because the key is still `Unit`).

If we change to:

```

LaunchedEffect(count) {
    println("Runs whenever count changes")
}

```

Now it runs again each time `count` changes.

2 What is `rememberCoroutineScope`

`rememberCoroutineScope()` gives you a **CoroutineScope** that is tied to the Composable lifecycle. You can use it to manually launch coroutines, usually in response to **user actions** (like button clicks).

```

val coroutineScope = rememberCoroutineScope()

Button(onClick = {
    coroutineScope.launch {
        // do some suspend work here
    }
}) {
    Text("Click me")
}

```

- Does **not** run automatically — you decide when to launch.
- Great for user-triggered work (e.g., refresh, save button).

- ⚠️ If you call `coroutineScope.launch` directly at the **top level** of a Composable, it will run **every recomposition** → BAD.

⚠️ Why you shouldn't launch directly

```
val coroutineScope = rememberCoroutineScope()

// ❌ BAD: Will run on every recomposition
coroutineScope.launch { doSomething() }
```

- Composables recompose often → this will trigger multiple network calls or tasks.

If you want something to run automatically once, **use `LaunchedEffect` instead**.

3 When to Use Each

Scenario	Best Choice
Run something once when the screen appears	✅ <code>LaunchedEffect(Unit)</code>
Run again whenever some state changes	✅ <code>LaunchedEffect(state)</code>
Trigger a job on user action (button click)	✅ <code>rememberCoroutineScope()</code>
Background tasks tied to ViewModel lifecycle	Use <code>viewModelScope</code>

4 Full Example — Fetch Public IP Using OkHttp + LaunchedEffect

```
// build.gradle (app level)
dependencies {
    implementation("androidx.compose.ui:ui:1.6.7")
    implementation("androidx.compose.material3:material3:1.2.1")
    implementation("androidx.activity:activity-compose:1.8.2")
    implementation("com.squareup.okhttp3:okhttp:4.12.0")
}

// MainActivity.kt
package com.example.ipfetcher

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
```

```

import androidx.compose.ui.unit.dp
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import okhttp3.OkHttpClient
import okhttp3.Request
import org.json.JSONObject

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme {
                Surface(modifier = Modifier.fillMaxSize()) {
                    IPAddressScreen()
                }
            }
        }
    }
}

@Composable
fun IPAddressScreen() {
    var ipAddress by remember { mutableStateOf("Fetching...") }

    // ✅ Runs only once when this Composable first appears
    LaunchedEffect(Unit) {
        ipAddress = fetchPublicIP()
    }

    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Text(text = ipAddress, style = MaterialTheme.typography.headlineMedium)
    }
}

suspend fun fetchPublicIP(): String = withContext(Dispatchers.IO) {
    try {
        val client = OkHttpClient()
        val request = Request.Builder()
            .url("https://api.ipify.org?format=json")
            .build()
        val response = client.newCall(request).execute()
        val body = response.body?.string()
        response.close()
        if (body != null) {
            val json = JSONObject(body)
            json.getString("ip")
        } else {
            "No response"
        }
    } catch (e: Exception) {
        "Error: ${e.message}"
    }
}

```

```
}  
}
```





- The coroutine runs **once automatically** because of `LaunchedEffect(Unit)` .
- If the Composable recomposes (e.g., due to state update), it will **not refetch** the IP.

5 If You Wanted a Manual Refresh (using `rememberCoroutineScope`)

```
@Composable  
fun IPAddressScreen() {  
    var ipAddress by remember { mutableStateOf("Tap to fetch IP") }  
    val coroutineScope = rememberCoroutineScope()  
  
    Column(  
        modifier = Modifier.fillMaxSize(),  
        horizontalAlignment = Alignment.CenterHorizontally,  
        verticalArrangement = Arrangement.Center  
    ) {  
        Text(text = ipAddress, style = MaterialTheme.typography.headlineMedium)  
        Spacer(modifier = Modifier.height(16.dp))  
        Button(onClick = {  
            coroutineScope.launch {  
                ipAddress = fetchPublicIP()  
            }  
        }) {  
            Text("Refresh")  
        }  
    }  
}
```

- Here, the network call only happens **when the user taps the button**.

Key Takeaways

-  **LaunchedEffect** = automatic, runs once when composed or when keys change. Safe for one-time side effects.
-  **rememberCoroutineScope** = manual; use for user-triggered coroutines.
-  Don't put `coroutineScope.launch { }` directly in the Composable body (it'll run on every recomposition).
-  Both cancel automatically when the Composable leaves the screen, keeping things memory-safe.

