1. AEM Backend (Sling Model and HTL for the Container Component):

- Create a Container Component in AEM: This component will act as the placeholder where other components can be dragged and dropped.
- Sling Model (`.java`): Create a Sling Model for your container component. This model will be responsible for retrieving the child components that have been added to it in the AEM Editor. You'll typically use the `@ChildResource` annotation to get a list of the child resources.

Java

```java
import java.util.List;
import javax.annotation.PostConstruct;
import javax.inject.Inject;
import javax.inject.Named;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.injectorspecific.ChildResource;

@Model(adaptables = Resource.class)
public class MyContainerModel {

    @Inject
    @Named(ResourceResolver.SYSTEM_USER_ID)
    private ResourceResolver resourceResolver;

    @ChildResource
    private List<Resource> items;

    private List<ComponentExporter> children;

    @PostConstruct
    protected void init() {
        children = new ArrayList<>();
        if (items != null) {
            for (Resource item : items) {
                ComponentExporter exporter = item.adaptTo(ComponentExporter.class);
                if (exporter != null) {
                    children.add(exporter);
                }
            }
        }
    }

    public List<ComponentExporter> getChildren() {
        return children;
    }
}
```

- HTL (`.html`): The HTL for your container component will primarily iterate through the child components retrieved by the Sling Model and use the `data-sly-resource` attribute to include their corresponding WCM components.

  HTML

  ```
  <div data-sly-use.model="com.example.aem.models.MyContainerModel">
      <div data-sly-list="${model.children}" data-sly-as="item">
          <div data-sly-resource="${item @ decoration=true}"></div>
      </div>
  </div>
  ```

  - `data-sly-use.model` : Instantiates your Sling Model.
  - `data-sly-list` : Iterates over the `children` list from the Sling Model.

  - `data-sly-resource` : Includes the WCM component associated with each child resource. The `@ decoration=true` attribute ensures that the necessary wrapper elements for the AEM Editor (like drag handles) are included.
- `_cq_dialog/.content.xml` : Define the edit dialog for your container component. This dialog will typically only have a tab for styling and potentially other container-specific properties. The key here is that you are *not* defining any specific child components within this dialog. The AEM Editor's drag-and-drop mechanism will handle adding components.
- `_content.xml` : Define the basic structure and properties of your container component. Ensure it's configured to be a container in the AEM Editor.

2. AEM SPA Editor SDK (Angular Component):

- Create an Angular Component: Create a corresponding Angular component that will represent your AEM container component in the SPA.

TypeScript

```
import { Component, Input, OnInit } from '@angular/core';
import { AEMAllowedComponents, MapTo, Container, ContainerItem, ContainerConfig } from

const EditConfig: ContainerConfig = {
  emptyLabel: 'Drag components here',
  dragDropName: 'My Container'
};

@Component({
  selector: 'app-my-container',
  template: `
    <div class="my-container" [cqPath]="cqPath" [cqItems]="items" [attr.data-cq-data-p
      <div *ngFor="let item of items">
```

```
        <cq-placeholder [path]="item.path" [attr.data-cq-data-path]="item.path"></cq-p
      </div>
    </div>
  `,
  styleUrls: ['./my-container.component.scss']
})
@MapTo('your-aem-project/components/content/my-container') // Replace with your compon
@AEMAllowedComponents({
  components: ['your-aem-project/components/content/text', 'your-aem-project/component
})
export class MyContainerComponent implements OnInit, Container {
  @Input() cqPath: string;
  @Input() cqItems: { [key: string]: ContainerItem };
  @Input() cqItemsOrder: string[];

  public items: ContainerItem[] = [];

  constructor() { }

  ngOnInit(): void {
    this.items = this.cqItemsOrder.map(key => this.cqItems[key]);
  }

  get containerEditConfig() {
    return EditConfig;
  }
}
```

- `@adobe/cq-angular-editable-components` : Import necessary modules from the AEM SPA Editor SDK.
- `@MapTo('your-aem-project/components/content/my-container')` : Maps your Angular component to the corresponding AEM component resource type. Replace this with the

actual resource type of your container component.

- `@AEMAllowedComponents` : This decorator specifies which AEM components can be dragged and dropped into this container in the AEM Editor. Provide an array of the resource types of the allowed components.
- `@Input() cqPath` , `@Input() cqItems` , `@Input() cqItemsOrder` : These inputs receive the necessary data from the AEM backend, including the path of the container and its child items.
- `cq-placeholder` : This directive from the AEM SPA Editor SDK is crucial. It acts as the rendering point for each child component within the container. The `[path]` attribute of the `cq-placeholder` tells the SDK which component to render in that slot.
- `containerEditConfig` : This getter returns the `EditConfig` object, which provides labels and drag-and-drop behavior hints in the AEM Editor.
- Template: The template iterates through the `items` and renders a `cq-placeholder` for each. The `[cqPath]` , `[cqItems]` , and `[attr.data-cq-data-path]` attributes are essential for the AEM Editor to understand and interact with the component.

3. Enabling the SPA Editor:

- Ensure your Angular application is integrated with the AEM SPA Editor. This typically involves setting up the `index.html` with the necessary `<cq-root>` element and configuring the `AppRoutingModule` to use the `RouterModule.forRoot(routes, { useHash: true })` .

How it Works:

1. Authoring in AEM Editor: When an author edits a page containing your container component in the AEM Editor, they will see the "Drag components here" placeholder (defined in `EditConfig`).

2. Drag and Drop: The author can drag components (that are allowed by the `@AEMAllowedComponents` decorator) into this container.
3. AEM Backend Updates: When a component is dragged and dropped, AEM updates the JCR structure of the container component by adding a new child resource for the dropped component.
4. SPA Request: When the SPA requests the JSON representation of the page, the Sling Model of your container component will fetch these child resources.
5. JSON Delivery: The Sling Model will adapt each child resource to its corresponding `ComponentExporter` (which is automatically handled by the AEM SPA Editor SDK if your components are properly implemented with `@Model(adaptables = Resource.class, exporters = { ComponentExporter.class })`). The JSON response will contain an object representing the container, with an array of its child components and their data.
6. Angular Rendering: Your Angular container component receives this JSON data through its `@Input()` properties (`cqPath`, `cqItems`, `cqItemsOrder`). It then iterates through the `cqItemsOrder` and renders a `cq-placeholder` for each item. The AEM SPA Editor SDK, based on the `path` property of each item, will dynamically render the corresponding Angular component that is mapped to the resource type of the child component.

Key Considerations:

- Component Mapping: Ensure that all the AEM components you want to be able to drag and drop into the container have corresponding Angular components and are correctly mapped using `@MapTo`.
- `@AEMAllowedComponents`: Carefully define the allowed components to maintain control over what authors can add to the container.

- Styling: You'll need to provide CSS styles for your container component and potentially for the layout of the child components within it. You can use the `cqPath` and `cqItems` properties to apply specific styles if needed.
- Responsive Grid within the Container: If you need a responsive grid *within* the container itself (allowing authors to arrange components in a grid layout inside the container), you would typically use the standard AEM `ResponsiveGrid` component as one of the allowed components that can be dragged into your custom container. The SPA Editor SDK handles the rendering of the `ResponsiveGrid` in the Angular SPA.

By following these steps, you can create an AEM component in your Angular SPA that functions as a container, allowing authors to drag and drop other AEM components inside it using the AEM Editor's familiar interface. The AEM SPA Editor SDK handles the communication between the AEM backend and your Angular frontend to make this seamless.

@adobe/cq-angular-editable-components is the old version. The new version was changed to @adobe/aem-angular-editable-components. How to use the new aem…

You are absolutely correct! `@adobe/cq-angular-editable-components` is indeed the older library. The current and recommended library is `@adobe/aem-angular-editable-components`.

The way you handle dynamic component rendering and the concept of a container have evolved in the newer library. The `cq-placeholder` directive is no longer the primary mechanism. Instead, you leverage the `AemComponentDirective` (often aliased as `aemComponent`) and the `[aemContext]` input binding.

Here's how you can achieve the container functionality using `@adobe/aem-angular-editable-components` :

1. AEM Backend (Sling Model and HTL for the Container Component):

The backend part remains largely the same as described before:

- Create a Container Component in AEM.
- Sling Model (`.java`): Retrieves the list of child resources that are `ComponentExporter` s.
- HTL (`.html`): Iterates through the child components and uses `data-sly-resource` with `@ decoration=true` .
- `_cq_dialog/.content.xml` : Defines the edit dialog (primarily for styling).
- `_content.xml` : Defines the component properties.

2. AEM SPA Editor SDK (Angular Component using `@adobe/aem-angular-editable-components`):

TypeScript

```
import { Component, Input, OnInit, ViewContainerRef, OnDestroy } from '@angular/core';
import {
  MapTo,
  Container,
  ContainerItem,
  ContainerConfig,
  AemComponentDirective,
  AemComponent,
  AemContext,
} from '@adobe/aem-angular-editable-components';
import { Subscription } from 'rxjs';
```

```
const EditConfig: ContainerConfig = {
  emptyLabel: 'Drag components here',
  dragDropName: 'My New Container',
};

@Component({
  selector: 'app-my-new-container',
  template: `
    <div class="my-new-container" [cqPath]="aemContext.path" [attr.data-cq-data-path]="aem
      <ng-container *ngFor="let item of items">
        <div [aemComponent]="item" [aemContext]="aemContext"></div>
      </ng-container>
      <div *ngIf="!items || items.length === 0" class="cq-placeholder" [attr.data-emptytex
    </div>
  `,
  styleUrls: ['./my-new-container.component.scss'],
})
@MapTo('your-aem-project/components/content/my-new-container') // Replace with your compon
@AemComponent({
  allowedComponents: ['your-aem-project/components/content/text', 'your-aem-project/compon
})
export class MyNewContainerComponent implements OnInit, OnDestroy, Container, AemComponent
  @Input() cqPath: string;
  @Input() cqItems: { [key: string]: ContainerItem };
  @Input() cqItemsOrder: string[];
  @Input() aemContext: AemContext;

  public items: ContainerItem[] = [];
  private subscription: Subscription;

  constructor(private viewContainerRef: ViewContainerRef) {
    this.aemContext = new AemContext(null, this.viewContainerRef);
  }

  ngOnInit(): void {
    this.subscription = this.aemContext.subscribe(() => {
      this.items = this.cqItemsOrder ? this.cqItemsOrder.map((key) => this.cqItems[key]) :
    });
  }

  ngOnDestroy(): void {
    if (this.subscription) {
      this.subscription.unsubscribe();
    }
  }

  get containerEditConfig() {
    return EditConfig;
  }
}
```

Key Changes and Explanations:

- `@adobe/aem-angular-editable-components` Import: You are now importing from the new library.
- `AemComponentDirective` (`[aemComponent]`): This directive is the replacement for `cq-placeholder` . You apply it to an HTML element (in this case, a `div`) within the `*ngFor` loop. The directive dynamically renders the Angular component that is mapped to the AEM component represented by the `item` .
- `AemContext` : This class provides contextual information about the current AEM component and its environment. You inject or create an instance of `AemContext` .
  - The `@Input() aemContext: AemContext;` receives the context from the parent (usually the page or another container).
  - In the constructor, we initialize `this.aemContext` for the component itself.

  - The `[aemContext]="aemContext"` binding passes the context down to the dynamically rendered child components.
- `AemComponent` Decorator: The `@AemComponent` decorator (imported from the new library) is used instead of `@AEMAllowedComponents` . It serves a similar purpose, allowing you to specify the `allowedComponents` .
- Template Structure:
  - The outer `div` has `[cqPath]="aemContext.path"` and `[attr.data-cq-data-path]="aemContext.path"` .
  - The `*ngFor` iterates through the `items` .
  - Inside the loop, the `div` with `[aemComponent]="item"` and `[aemContext]="aemContext"` is where the dynamic rendering happens. The `item` here represents the data for each child component.
  - The `*ngIf` block displays the `cq-placeholder` (with the `emptyLabel`) when there are no items in the container, providing the drag-and-drop target in the AEM Editor. Note that this `cq-placeholder` is a standard HTML element with specific attributes, not the old Angular directive.
- `OnInit` and Subscription: The component subscribes to changes in the `aemContext` . When the context updates (e.g., due to changes in the AEM Editor), the `items` array is recalculated based on `cqItemsOrder` and `cqItems` . This ensures the view updates correctly.
- `OnDestroy` : It's important to unsubscribe from the `aemContext` subscription to prevent memory leaks.

How it Works with the New Library:

1. Authoring in AEM Editor: The author drags and drops components into the container.

2. AEM Backend Updates: The JCR structure is updated.
3. SPA Request: The SPA requests the JSON representation.
4. JSON Delivery: The Sling Model provides the data for the container, including an array of its child components with their resource types and properties.
5. Angular Rendering:
   - Your `MyNewContainerComponent` receives this data through the `@Input()` bindings.
   - The `*ngFor` loop iterates over the `items` .
   - For each `item` , the `[aemComponent]="item"` directive dynamically looks up the Angular component that is registered (via `@MapTo`) for the resource type of that `item` .
   - The `[aemContext]="aemContext"` binding provides the necessary AEM context to the dynamically rendered child components, allowing them to be editable within the AEM Editor.

Important Considerations:

- Component Mapping (`@MapTo`): Ensure all your draggable AEM components have corresponding Angular components and are correctly mapped using `@MapTo` from `@adobe/aem-angular-editable-components` .
- `allowedComponents` : Use the `allowedComponents` array in the `@AemComponent` decorator to control which components can be added to the container.
- `AemContext` : Understand the role of `AemContext` in providing the necessary information for editable components.
- Styling: Apply CSS styles to your container and its children as needed.

By migrating to `@adobe/aem-angular-editable-components` and using the `AemComponentDirective` with `[aemContext]`, you can effectively create container components in your AEM SPA that support drag-and-drop functionality in the AEM Editor, following the modern approach of the SDK.