# Migrate to React Flow 12

You can find the docs for old versions of React Flow here: v11, v10, v9

Before you can use the new features that come with React Flow 12 like server side rendering, computing flows, and dark mode, here are the breaking changes you'll have to address first. We tried to keep the breaking changes to a minimum, but some of them were necessary to implement the new features.

## Migration guide

Before you start to migrate, you need to install the new package.

npm pnpm yarn bun

```
npm install @xyflow/react
```

### 1. A new npm package name

The package `reactflow` has been renamed to `@xyflow/react` and it's not a default import anymore. You also need to adjust the style import. Before v12, React Flow was divided into multiple packages. That's not the case anymore. If you just used the core, you now need to install the `@xyflow/react` package.

**Old API**

```
// npm install reactflow
import ReactFlow from 'reactflow';
```

**New API**

```
// npm install @xyflow/react
import { ReactFlow } from '@xyflow/react';

// you also need to adjust the style import
import '@xyflow/react/dist/style.css';

// or if you just want basic styles
import '@xyflow/react/dist/base.css';
```

## 2. Node `measured` attribute for measured `width` and `height`

All measured node values are now stored in `node.measured`. Besides the new package name, this is the biggest change. After React Flow measures your nodes, it writes the dimesions to `node.measured.width` and `node.measured.height`. If you are using any layouting library like dagre or elk, you now need to take the dimesions from `node.measured` instead of `node`. If you are using `width` and `height`, those values will now be used as inline styles to specify the node dimensions.

**Old API**

```
// getting the measured width and height
const nodeWidth = node.width;
const nodeHeight = node.height;
```

**New API**

```
// getting the measured width and height
const nodeWidth = node.measured?.width;
const nodeHeight = node.measured?.height;
```

## 3. New dimension handling `node.width` / `node.height` vs `node.measured.width` / `node.measured.height`

In order to support server side rendering we had to restructure the API a bit, so that users can pass node dimensions more easily. For this we changed the behaviour of the `node.width` and `node.height` attributes. In React Flow 11, those attributes were measured values and only used as a reference. In React Flow 12 those attributes are used as inline styles to specify the node dimensions. If you load nodes from a database, you probably want to remove the `width` and `height` attributes from your nodes, because the behaviour is slightly different now. Using `width` and `height` now means that the dimensions are not dynamic based on the content but fixed.

### Old API

```
// in React Flow 11 you might used node.style to set the dimensions
const nodes = [
  {
    id: '1',
    type: 'input',
    data: { label: 'input node' },
    position: { x: 250, y: 5 },
    style: { width: 180, height: 40 },
  },
];
```

### New API

```
// in React Flow 12 you can used node.width and node.height to set the dimensi
const nodes = [
  {
    id: '1',
    type: 'input',
    data: { label: 'input node' },
    position: { x: 250, y: 5 },
    width: 180,
    height: 40,
  },
];
```

If you want to read more about how to configure React Flow for server side rendering, you can read about it in the server side rendering guide.

## 4. Updating nodes and edges

We are not supporting node and edge updates with object mutations anymore. If you want to update a certain attribute, you need to create a new node / edge.

### Old API

```
setNodes((currentNodes) =>
  currentNodes.map((node) => {
    node.hidden = true;
    return node;
  }),
);
```

### New API

```
setNodes((currentNodes) =>
  currentNodes.map((node) => ({
    ...node,
    hidden: true,
  })),
);
```

## 5. Rename `onEdgeUpdate` (and related APIs) to `onReconnect`

We renamed the `onEdgeUpdate` function to `onReconnect` and all related APIs (mentioned below). The new name is more descriptive and makes it clear that the function is used to reconnect edges.

- `updateEdge` renamed to `reconnectEdge`
- `onEdgeUpdateStart` renamed to `onReconnectStart`
- `onEdgeUpdate` renamed to `onReconnect`
- `onEdgeUpdateEnd` renamed to `onReconnectEnd`
- `edgeUpdaterRadius` renamed to `reconnectRadius`
- `edge.updatable` renamed to `edge.reconnectable`
- `edgesUpdatable` renamed to `edgesReconnectable`

### Old API

```
<ReactFlow
  onEdgeUpdate={onEdgeUpdate}
```

```
    onEdgeUpdateStart={onEdgeUpdateStart}
    onEdgeUpdateEnd={onEdgeUpdateEnd}
  />
```

### New API

```
  <ReactFlow
    onReconnect={onReconnect}
    onReconnectStart={onReconnectStart}
    onReconnectEnd={onReconnectEnd}
  />
```

## 6. Rename `parentNode` to `parentId`

If you are working with subflows, you need to rename `node.parentNode` to `node.parentId`. The `parentNode` attribute was a bit misleading, because it was not a reference to the parent node, but the `id` of the parent node.

### Old API

```
  const nodes = [
    // some nodes ...
    {
      id: 'xyz-id',
      position: { x: 0, y: 0 },
      type: 'default',
      data: {},
      parentNode: 'abc-id',
    },
  ];
```

### New API

```
  const nodes = [
    // some nodes ...
    {
      id: 'xyz-id',
      position: { x: 0, y: 0 },
      type: 'default',
      data: {},
      parentId: 'abc-id',
    },
  ];
```

## 7. Custom node props

We renamed the `xPos` and `yPos` props to `positionAbsoluteX` and `positionAbsoluteY`

### Old API

```
function CustomNode({ xPos, yPos }) {
  ...
}
```

### New API

```
function CustomNode({ positionAbsoluteX, positionAbsoluteY }) {
  ...
}
```

## 8. Handle component class names

We renamed some of the classes used to define the current state of a handle.

- `react-flow__handle-connecting` renamed to `connectingto` / `connectingfrom`
- `react-flow__handle-valid` renamed to `valid`

## 9. `getNodesBounds` options

The type of the second param changed from `nodeOrigin` to `options.nodeOrigin`

### Old API

```
const bounds = getNodesBounds(nodes: Node[], nodeOrigin)
```

### New API

```
const bounds = getNodesBounds(nodes: Node[], { nodeOrigin })
```

## 10. Typescript changes for defining nodes and edges

We simplified types and fixed issues about functions where users could pass a NodeData generic. The new way is to define your own node type with a union of all your nodes. With this change, you can now have multiple node types with different data structures and always be able to distinguish by checking the `node.type` attribute.

**New API**

```
type NumberNode = Node<{ value: number }, 'number'>;
type TextNode = Node<{ text: string }, 'text'>;
type AppNode = NumberNode | TextNode;
```

You can then use the `AppNode` type as the following:

```
const nodes: AppNode[] = [
  { id: '1', type: 'number', data: { value: 1 }, position: { x: 100, y: 100 }
  { id: '2', type: 'text', data: { text: 'Hello' }, position: { x: 200, y: 200
];
```

```
const onNodesChange: onNodesChange<AppNode> = useCallback((changes) => setNode
```

You can read more about this in the Typescript guide.

## 11. Rename `nodeInternals`

If you are using `nodeInternals` you need to rename it to `nodeLookup`.

**Old API**

```
const node = useStore((s) => s.nodeInternals.get(id));
```

New API

```
const node = useStore((s) => s.nodeLookup.get(id));
```

### 12. Removal of deprecated functions

We removed the following deprecated functions:

- `getTransformForBounds` (replaced by `getViewportForBounds`)
- `getRectOfNodes` (replaced by `getNodesBounds`)
- `project` (replaced by `screenToFlowPosition`)
- `getMarkerEndId`
- `updateEdge` (replaced by `reconnectEdge`)

### 13. Custom `applyNodeChanges` and `applyEdgeChanges`

If you wrote your own function for applying changes, you need to handle the new "replace" event. We removed the "reset" event and added a "replace" event that replaces specific nodes or edges.

## New features

Now that you successfully migrated to v12, you can use all the fancy features. As mentioned above, the biggest updates for v12 are:

### 1. Server side rendering

You can define `width`, `height` and `handles` for the nodes. This makes it possible to render a flow on the server and hydrate on the client: server side rendering guide.

- **Details:** In v11, `width` and `height` were set by the library as soon as the nodes got measured. This still happens, but we are now using `measured.width` and `measured.height` to store this information. In the previous versions there was always a lot of confusion about `width` and `height`. It's hard to understand, that you can't use it for passing an actual width or height. It's also not obvious that those attributes get added by the library. We think that the new implementation solves both of the problems: `width` and `height` are optional attributes that can be used to define dimensions and everything that is set by the library, is stored in `measured`.

## 2. Computing flows

The new hooks `useHandleConnections` and `useNodesData` and the new `updateNode` and `updateNodeData` functions (both are part of `useReactFlow`) can be used to manage the data flow between your nodes: computing flows guide. We also added those helpers for edges (`updateEdge` and `updateEdgeData`)!

- **Details:** Working with flows where one node data relies on another node is super common. You update node A and want to react on those changes in the connected node B. Until now everyone had to come up with a custom solution. With this version we want to change this and give you performant helpers to handle use cases like this.

## 3. Dark mode and CSS variables

React Flow now comes with a built-in dark mode, that can be toggled by using the new `colorMode` prop (”light”, “dark” or “system”): dark mode example

- **Details:** With this version we want to make it easier to switch between dark and light modes and give you a better starting point for dark flows. If you pass `colorMode="dark"`, we add the class name “dark” to the wrapper and use it to adjust the styling. To make the implementation for this new feature easier on our ends, we switched to CSS variables for most of the styles. These variables can also be used in user land to customize a flow.

## 4. A better DX with TSDoc

We started to use TSDoc for a better DX. While developing your IDE will now show you the documentation for the props and hooks. This is a big step for us to make the library more accessible and easier to use. We will also use TSDoc in the near future to generate the documentation.

## More features and updates

There is more! Besides the new main features, we added some minor things that were on our list for a long time:

- `useConnection` **hook**: With this hook you can access the ongoing connection. For example, you can use it for colorizing handles styling a custom connection line based on the current start / end handles.
- **Controlled `viewport`**: This is an advanced feature. Possible use cases are to animate the viewport or round the transform for lower res screens for example. This features brings two new props: `viewport` and `onViewportChange`.
- `ViewportPortal` **component:** This makes it possible to render elements in the viewport without the need to implement a custom node.
- `onDelete` **handler**: We added a combined handler for `onDeleteNodes` and `onDeleteEdges` to make it easier to react to deletions.
- `onBeforeDelete` **handler**: With this handler you can prevent/ manage deletions.
- `isValidConnection` **prop:** This makes it possible to implement one validation function for all connections. It also gets called for programmatically added edges.
- `autoPanSpeed` **prop:** For controlling the speed while auto panning.
- `paneClickDistance` **prop:** max distance between mousedown/up that will trigger a click.
- **Background component**: add `patternClassName` prop to be able to style the background pattern by using a class name. This is useful if you want to style the background pattern with Tailwind for example.
- **`onMove` callback** gets triggered for library-invoked viewport updates (like fitView or zoom-in)
- **`deleteElements`** now returns deleted nodes and deleted edges
- add **`origin` attribute** for nodes
- add **`selectable` attribute** for edges
- **Node Resizer updates**: child nodes don't move when the group is resized, extent and expand is recognized correctly
- Correct types for `BezierEdge`, `StepEdge`, `SmoothStepEdge` and `StraightEdge` components
- New edges created by the library only have `sourceHandle` and `targetHandle` attributes when those attributes are set. (We used to pass `sourceHandle: null` and `targetHandle: null`)
- Edges do not mount/unmount when their z-index change
- connection line knows about the target handle position so that the path is drawn correctly
- `nodeDragThreshold` is 1 by default instead of 0

- a better selection box usability (capture while dragging out of the flow)
- add `selectable`, `deletable`, `draggable` and `parentId` to `NodeProps`
- add a warning when styles not loaded

## Internal changes

These changes are not really user-facing, but it could be important for folks who are working with the internal React Flow store:

- The biggest internal change is that we created a new package **@xyflow/system with framework agnostic helpers** that can be used by React Flow and Svelte Flow
  - **XYDrag** for handling dragging node(s) and selection
  - **XYPanZoom** for controlling the viewport panning and zooming
  - **XYHandle** for managing new connections
- We renamed `nodeInternals` to `nodeLookup`. That map serves as a lookup, but we are not creating a new map object on any change so it's really only useful as a lookup.
- We removed the internal "reset" event and added a "replace" event to be able to update specific nodes.
- We removed `connectionNodeId`, `connectionHandleId`, `connectionHandleType` from the store and added `connection.fromHandle.nodeId`, `connection.fromHandle.id`, …
- add `data-id` to edges
- `onNodeDragStart`, `onNodeDrag` and `onNodeDragStop` also get called when user drags a selection (in addition to `onSelectionDragStart`, `onSelectionDrag`, `onSelectionDragStop`)

Last updated on December 4, 2024

Getting Started
API Reference
Examples
Showcase

Discord
Github
X / Twitter
Bluesky

xyflow

Blog
Open Source
About
Contact
Careers hiring

Legal

MIT License
Code of Conduct
Imprint