Bootstrap's default grid system represents the culmination of over a decade of CSS layout techniques, tried and tested by millions of people. But, it was also created without many of the modern CSS features and techniques we're seeing in browsers like the new CSS Grid.

{{< callout warning >}} **Heads up—our CSS Grid system is experimental and opt-in as of v5.1.0!** We included it in our documentation's CSS to demonstrate it for you, but it's disabled by default. Keep reading to learn how to enable it in your projects. {{< /callout >}}

How it works

With Bootstrap 5, we've added the option to enable a separate grid system that's built on CSS Grid, but with a Bootstrap twist. You still get classes you can apply on a whim to build responsive layouts, but with a different approach under the hood.

- **CSS Grid is opt-in.** Disable the default grid system by setting `$enable-grid-classes: false` and enable the CSS Grid by setting `$enable-cssgrid: true`. Then, recompile your Sass.
- **Replace instances of `.row` with `.grid`.** The `.grid` class sets `display: grid` and creates a `grid-template` that you build on with your HTML.
- **Replace `.col-*` classes with `.g-col-*` classes.** This is because our CSS Grid columns use the `grid-column` property instead of `width`.
- **Columns and gutter sizes are set via CSS variables.** Set these on the parent `.grid` and customize however you want, inline or in a stylesheet, with `--bs-columns` and `--bs-gap`.

In the future, Bootstrap will likely shift to a hybrid solution as the `gap` property has achieved nearly full browser support for flexbox.

Key differences

Compared to the default grid system:

- Flex utilities don't affect the CSS Grid columns in the same way.
- Gaps replaces gutters. The `gap` property replaces the horizontal `padding` from our default grid system and functions more like `margin`.
- As such, unlike `.row`s, `.grid`s have no negative margins and margin utilities cannot be used to change the grid gutters. Grid gaps are applied horizontally and vertically by default. See the [customizing section] for more details.
- Inline and custom styles should be viewed as replacements for modifier classes (e.g., `style="--bs-columns: 3;"` vs `class="row-cols-3"`).
- Nesting works similarly, but may require you to reset your column counts on each instance of a nested `.grid`. See the [nesting section] for details.

Examples

Three columns

Three equal-width columns across all viewports and devices can be created by using the `.g-col-4` classes. Add [responsive classes]() to change the layout by viewport size.

{{< example class="bd-example-cssgrid" >}}

.g-col-4
.g-col-4
.g-col-4
{{< /example >}}

Responsive

Use responsive classes to adjust your layout across viewports. Here we start with two columns on the narrowest viewports, and then grow to three columns on medium viewports and above.

{{< example class="bd-example-cssgrid" >}}

.g-col-6 .g-col-md-4
.g-col-6 .g-col-md-4
.g-col-6 .g-col-md-4
{{< /example >}}
Compare that to this two column layout at all viewports.

{{< example class="bd-example-cssgrid" >}}

.g-col-6
.g-col-6
{{< /example >}}

Wrapping

Grid items automatically wrap to the next line when there's no more room horizontally. Note that the `gap` applies to horizontal and vertical gaps between grid items.

{{< example class="bd-example-cssgrid" >}}

.g-col-6
.g-col-6
.g-col-6
.g-col-6
{{< /example >}}

Starts

Start classes aim to replace our default grid's offset classes, but they're not entirely the same. CSS Grid creates a grid template through styles that tell browsers to "start at this column" and "end at this column." Those properties are `grid-column-start` and `grid-column-end`. Start classes are shorthand for the former. Pair them with the column classes to size and align your columns however you need. Start classes begin at `1` as `0` is an invalid value for these properties.

{{< example class="bd-example-cssgrid" >}}

.g-col-3 .g-start-2
.g-col-4 .g-start-6
{{< /example >}}

Auto columns

When there are no classes on the grid items (the immediate children of a `.grid`), each grid item will automatically be sized to one column.

{{< example class="bd-example-cssgrid" >}}

1
1
1
1
1
1
1
1
1
1
1
1
{{< /example >}}
This behavior can be mixed with grid column classes.

{{< example class="bd-example-cssgrid" >}}

.g-col-6
1
1
1
1
1
1
{{< /example >}}

Nesting

Similar to our default grid system, our CSS Grid allows for easy nesting of `.grid`s. However, unlike the default, this grid inherits changes in the rows, columns, and gaps. Consider the example below:

- We override the default number of columns with a local CSS variable: `--bs-columns: 3`.
- In the first auto-column, the column count is inherited and each column is one-third of the available width.
- In the second auto-column, we've reset the column count on the nested `.grid` to 12 (our default).
- The third auto-column has no nested content.

In practice this allows for more complex and custom layouts when compared to our default grid system.

{{< example class="bd-example-cssgrid" >}}

First auto-column
Auto-column
Auto-column
Second auto-column
6 of 12
4 of 12

2 of 12
Third auto-column
{{< /example >}}

Customizing

Customize the number of columns, the number of rows, and the width of the gaps with local CSS variables.

{{< bs-table "table" >}}

Variable	Fallback value	Description
`--bs-rows`	`1`	The number of rows in your grid template
`--bs-columns`	`12`	The number of columns in your grid template
`--bs-gap`	`1.5rem`	The size of the gap between columns (vertical and horizontal)
{{< /bs-table >}}		

These CSS variables have no default value; instead, they apply fallback values that are used *until* a local instance is provided. For example, we use `var(--bs-rows, 1)` for our CSS Grid rows, which ignores `--bs-rows` because that hasn't been set anywhere yet. Once it is, the `.grid` instance will use that value instead of the fallback value of `1`.

No grid classes

Immediate children elements of `.grid` are grid items, so they'll be sized without explicitly adding a `.g-col` class.

{{< example class="bd-example-cssgrid" >}}

Auto-column
Auto-column
Auto-column
{{< /example >}}

Columns and gaps

Adjust the number of columns and the gap.

{{< example class="bd-example-cssgrid" >}}

.g-col-2
.g-col-2
{{< /example >}}

{{< example class="bd-example-cssgrid" >}}

.g-col-6
.g-col-4
{{< /example >}}

Adding rows

Adding more rows and changing the placement of columns:

{{< example class="bd-example-cssgrid" >}}

Auto-column
Auto-column
Auto-column
{{< /example >}}

Gaps

Change the vertical gaps only by modifying the `row-gap`. Note that we use `gap` on `.grid`s, but `row-gap` and `column-gap` can be modified as needed.

{{< example class="bd-example-cssgrid" >}}

.g-col-6
.g-col-6
.g-col-6
.g-col-6
{{< /example >}}
Because of that, you can have different vertical and horizontal `gap`s, which can take a single value (all sides) or a pair of values (vertical and horizontal). This can be applied with an inline style for `gap`, or with our `--bs-gap` CSS variable.

{{< example class="bd-example-cssgrid" >}}

.g-col-6
.g-col-6
.g-col-6
.g-col-6
{{< /example >}}

Sass

One limitation of the CSS Grid is that our default classes are still generated by two Sass variables, `$grid-columns` and `$grid-gutter-width`. This effectively predetermines the number of classes generated in our compiled CSS. You have two options here:

- Modify those default Sass variables and recompile your CSS.
- Use inline or custom styles to augment the provided classes.

For example, you can increase the column count and change the gap size, and then size your "columns" with a mix of inline styles and predefined CSS Grid column classes (e.g., `.g-col-4`).

{{< example class="bd-example-cssgrid" >}}

14 columns
.g-col-4
{{< /example >}}