

◀ CODEPLY BLOG

📧 SUBSCRIBE

How the Bootstrap Grid Really Works

06 APRIL 2016 on responsive-design, bootstrap, grid, responsive-layouts, bootstrap-4

[Google +](#)

Bootstrap Grid Explained

Over the last few years I've answered a lot (over 800!) of [Bootstrap questions on Stack Overflow](#). Frequently these questions relate to [the Bootstrap grid](#) — that powerful blend of containers, rows and columns that make responsive layouts possible and flexible with Bootstrap.

More often than not, questions about the Bootstrap grid can be answered by understanding a few basics about what exactly the grid is, and how it works. I will clarify the Bootstrap grid here so that you're no longer left wondering about things like...

How do the different grid sizes (or breakpoints) work?

Why does Bootstrap have a 12 unit (columns) grid?

Can I have more than 12 columns in a single row?

Why does the row have a negative margin?

Grid Basics

It doesn't take much time before Bootstrap newbies feel at home with the grid. Here's the HTML markup we'd use in Bootstrap to make a simple 2-column Web page layout.

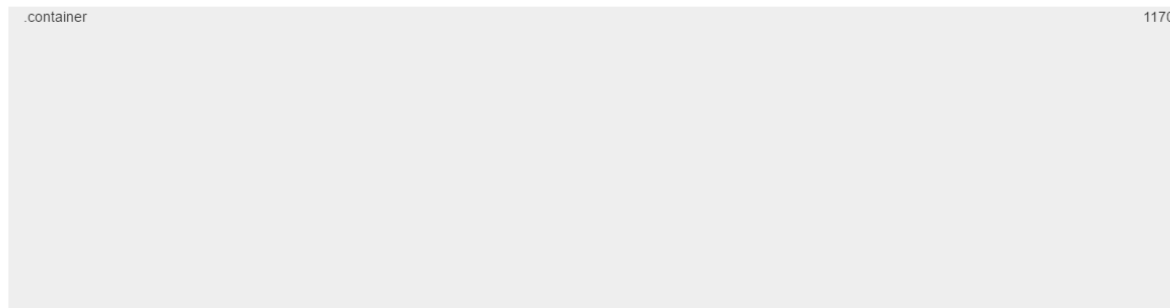
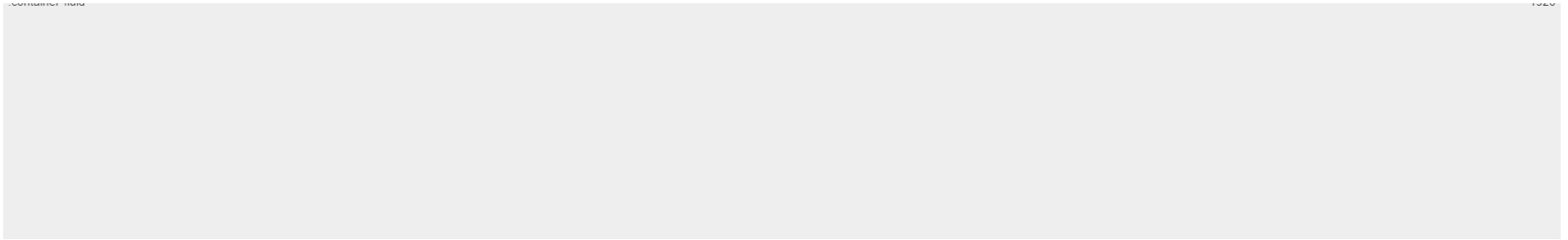
```
<div class="container">
  <div class="row">
    <div class="col-md-3">I am 3 units wide</div>
    <div class="col-md-9">I am 9 units wide</div>
  </div>
</div>
```

Containers

The root (top-level) element of the Bootstrap grid is the `container`. At first the container may seem trivial or unnecessary, but it's very important in controlling layout width.

Bootstrap provides 2 types (classes) of containers:

- `<div class="container-fluid">` - full-width container that spans the entire viewport width.
- `<div class="container">` - fixed-width container centered in the middle the viewport.



The `container` scales down in width responsively so that it eventually becomes 100% width (the same as `container-fluid`) on smaller devices.

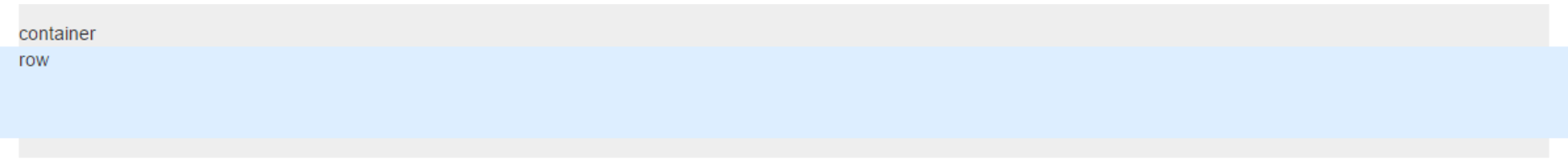
[Try the container demo](#)

Bootstrap containers (`container` or `container-fluid`) work in close partnership with the `row` and `col-*` (columns) to create "the grid".

Rows & Columns

Inside the container, the `row` class is used to contain the grid columns. Rows should *always*

be placed inside of a container to ensure proper spacing (between page content and the edge of the browser). If you don't put a `row` inside a container, the `row` will be wider than the width of the viewport, causing a horizontal scrollbar.



container
row

The Bootstrap `row` uses *negative margins* (-15px) to counteract the padding (15px) of the `container`. The end result is *no* visual spacing (margin or padding) on the sides of the `row` within the `container`. There are several reasons why the "negative margin" approach is used in Bootstrap.

Why does Bootstrap use negative margins?

The "negative margin" approach isn't just a gimmick, it works better. Some grid systems instead adjust the padding (or margin) on the first/last columns using CSS child selector logic. But, [as you can see here](#) this *doesn't* work well since the first & last columns rendered *visually*, can be different than the actual first & last columns in the HTML markup of any given row. There are also other reasons Bootstrap uses the negative margins including simpler CSS, nesting and column wrapping which is discussed later in this article.

The relationship between `container` and `row` is made complete with columns (`col-*`). Each `row` is divided horizontally using Bootstrap's column classes `col-*`. Only columns should be the immediate children of rows. Horizontal layout across the page is relevant because screen width is an important consideration for responsive design.

bootstrap 12 columns

The visual space between each column (known as the **gutter**) is created with padding (15px) on the left & right side of each column. As a result, the effective gutter between the content of each column is 30px (15px on the left + 15px on the right). The first (left-most) and last (right-most) columns line up perfectly to the edge of the `row` thanks to that *negative margin* discussed earlier.

Often users ask "Why does Bootstrap have 12 columns?". The reason Bootstrap has a 12-unit grid (instead of 10, 16, etc..) is that 12 evenly divides into 6 (halves), 4 (quarters) and 3 (thirds). This makes adapting to a variety of layouts much easier. Bootstrap's grid columns are identified by different `col-(breakpoint)-(units)` CSS classes. So for example, `col-md-3` would be a column that takes up 3 of the 12 units (or 25%) across in a `row`.

To enable or "set" a column width in your layout, simply use the appropriate `col-*-*` class in your HTML markup. The columns consume a percentage width of their container:

Column units & widths

col-md-1	=	8.3% width
col-md-2	=	16.7% width
col-md-3	=	25% width
col-md-4	=	33.3% width
col-md-5	=	41.6% width
col-md-6	=	50% width
col-md-7	=	58.3% width
col-md-8	=	66.8% width
col-md-9	=	75% width
col-md-10	=	83.3% width
col-md-11	=	91.6% width
col-md-12	=	100% width

Examples

2 columns: 25% on left, 75% on right

```
<div class="row">
  <div class="col-md-3">..</div>
  <div class="col-md-9">..</div>
</div>
```

2 columns: 50% on left, 50% on right

```
<div class="row">
  <div class="col-md-6">..</div>
  <div class="col-md-6">..</div>
</div>
```

3 columns: 33% each

```
<div class="row">
  <div class="col-md-4">..</div>
  <div class="col-md-4">..</div>
  <div class="col-md-4">..</div>
</div>
```

As you see, the appropriate `col-*` class is used to create various layouts.

Grid sizes (A.K.A - Breakpoints)

In addition to the concept of column width, Bootstrap has different "breakpoints" or grid sizes. The Bootstrap 3 grid comes in four (4) sizes to accomodate different screen (or viewport) widths. Each grid "size" encompasses a *range* that is designed to best-fit typical device screen widths such as that of desktops, laptops, tablets and smartphones. Bootstrap uses CSS media queries to create responsive breakpoints that establish a boundary for each grid size. These grid sizes enable you to change the layout of columns to best match different screen widths and devices__ the essence of responsive design.

- **xs** - for the *smallest* screen widths like smartphones < 768 px
- **sm** - for *small* screen widths like smartphones and tablets >= 768 px
- **md** - for *medium* screen widths like tablets and laptops >= 992 px
- **lg** - for *large* screen widths like desktops >= 1200 px

The upcoming Bootstrap 4 grid will add a new `xl` breakpoint to accomodate the widest high-res Desktop screens. The various breakpoints are combined with the column units to create different column layouts on different devices. For example, `col-md-3` would be 25% width on medium size screens, and then you could add `col-xs-6` to make the same column 50% width on the smallest screens. To enable or "use" a grid size, you simply specify it using the appropriate `col-*-*` class in your HTML markup. For example,

3-units wide on **medium** screens..

```
<div class="col-md-3">..</div>
```

6-units wide on small screens..

```
<div class="col-sm-6">..</div>
```

Combine the classes to use change column widths on different grid sizes..

```
<div class="col-md-3 col-sm-6">..</div>
```

Key points to remember about breakpoints and grid sizes:

- Columns will stack vertically (and become full-width) on `xs` screens unless you use a specific `col-xs-*` class in your HTML markup. Use `xs` to prevent vertical stacking.
- The smaller grid classes *also* apply to larger screens unless overridden specifically for larger screens. So, `<div class="col-md-6"></div>` is effectively the same as `<div class="col-md-6 col-lg-6"></div>`. Therefore, you only need to use the class for the smallest device width you want to support.

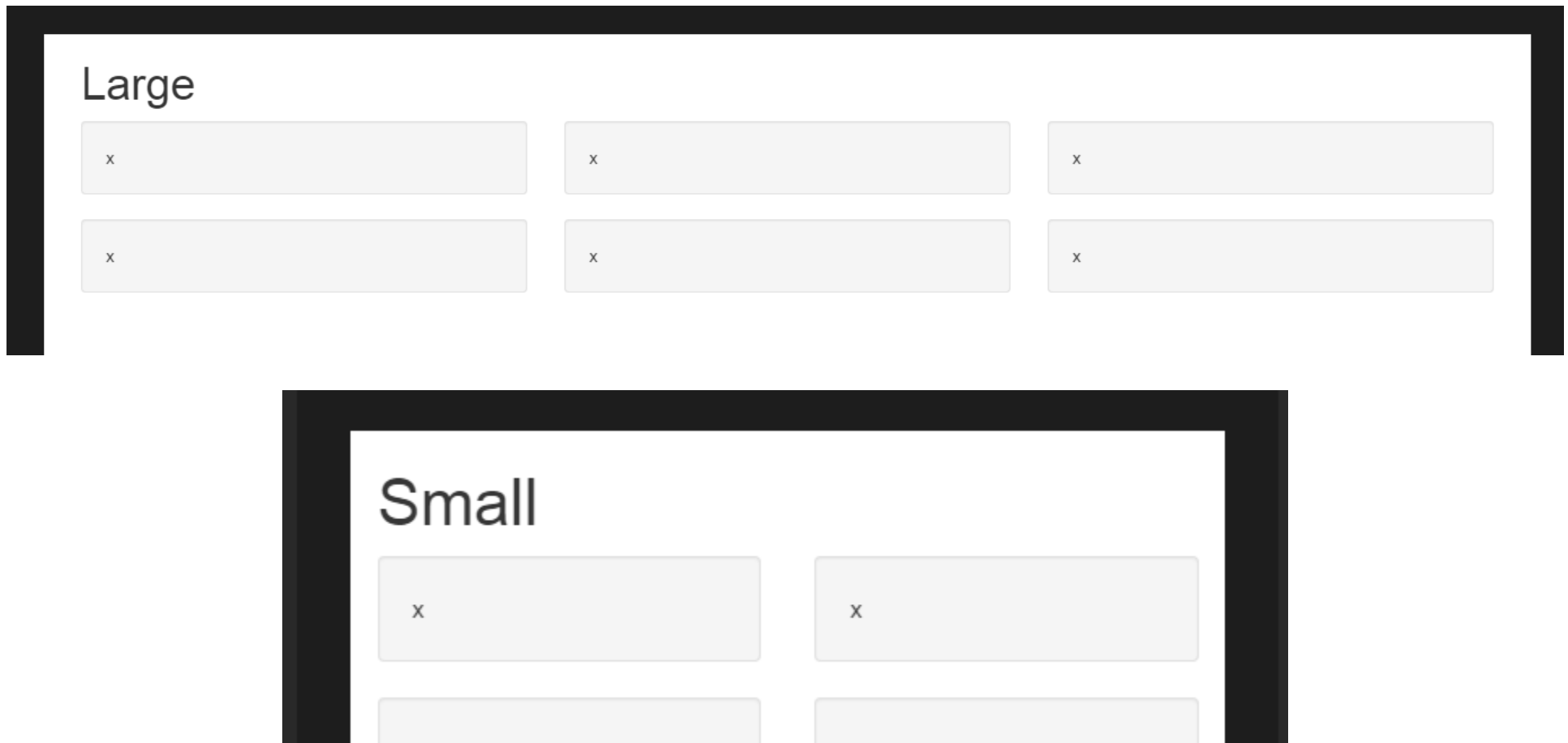
Here are [several more examples](#) that demonstrate various Bootstrap grid layouts using different breakpoints.

Advanced Bootstrap Layouts

As you create different responsive layouts you find that some scenarios require more complex combinations of rows & columns to make "things fit" on multiple devices.

Column Wrapping

In some responsive scenarios you'll see that it's necessary to have *columns exceeding 12 units* in a single `.row` element. Consider for example a layout where we want *3 columns on larger devices*, and *2 columns on smaller devices*:





To get this layout, our Bootstrap grid markup looks like:

```
<div class="row">
  <div class="col-xs-6 col-md-4">
    x
  </div>
  <div class="col-xs-6 col-md-4">
    x
  </div>
  <div class="col-xs-6 col-md-4">
    x
  </div>
  <div class="col-xs-6 col-md-4">
    x
  </div>
</div>
```

```
<div class="col-xs-6 col-md-4">  
  x  
</div>  
<div class="col-xs-6 col-md-4">  
  x  
</div>  
</div>
```

As you see in the markup, the total of column units in the row exceeds 12. This technique is known as column wrapping and it's one of Bootstrap's most powerful RWD features.

From the [Bootstrap docs](#):

"If more than 12 columns are placed within a single row, each group of extra columns will, as one unit, wrap onto a new line"

So, "yes, it's okay to have more than 12 units in a row". Just remember the wrapping is effected by column height and in some cases you need to use responsive resets to "clear" the wrapping content.

[Read More: When to Use Bootstrap's Row Class](#)

Offsets

The *offset* feature of the grid enables you to move columns to the right, or create more whitespace to the left. This comes in handy when you want a **narrower centered layout**. Here's an example of 2 - 33% width columns (`col-*-4`) that are centered by moving the first columns over 2 units (`col-offset-md-2`).

```
<div class="row">
  <div class="col-offset-md-2 col-md-4">
    1
  </div>
  <div class="col-md-4">
    2
  </div>
</div>
```

Nesting

Many scenarios require that *columns inside columns*, and in this case our columns become the containers. Bootstrap allows us to nest `row` and `col-*` inside other `col-*` which enables more control over when columns stack vertically at specific breakpoints. If we remember to think "mobile-first", we can nest the desired smaller device layout *inside* the desired larger layout. This enables the columns to switch positions (change the order) when the columns stack vertically on small devices:

```
<div class="row">
  <div class="col-md-6">
    <div class="row">
      <div class="col-md-12">
        1
      </div>
      <div class="col-md-12">
        2
      </div>
    </div>
  </div>
  <div class="col-md-6">
    3
  </div>
  <div class="col-md-6">
    4
  </div>
</div>
```

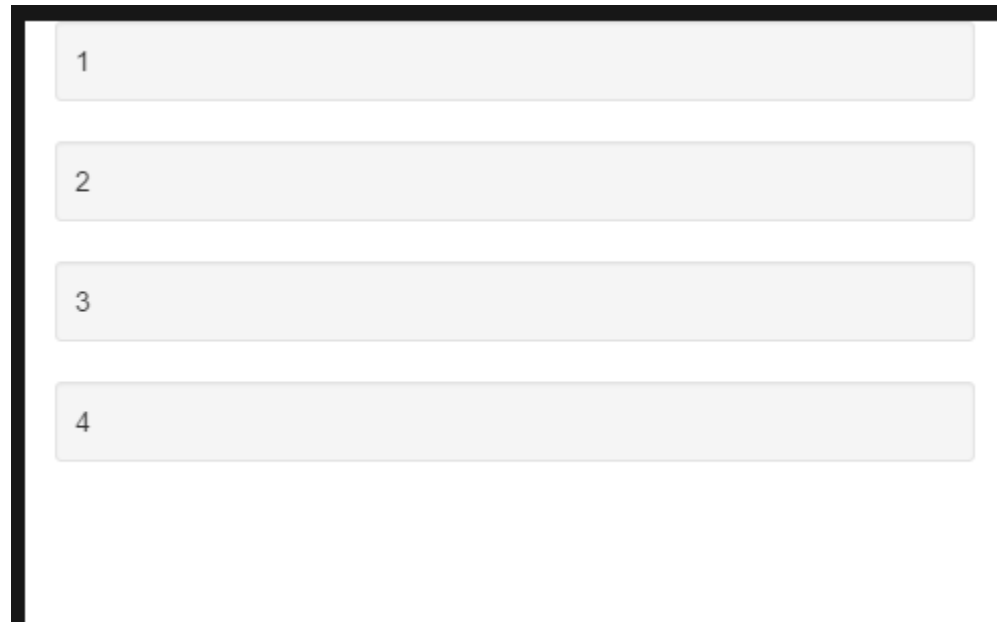
Large/medium layout (Note the column order):



2

4

Small layout:



When nesting, always remember to use another `row` around the inner columns to ensure proper spacing/padding. Also inner rows shouldn't exceed 12 columns. While you can use nesting to change the position of columns on different devices, you can also use Bootstrap's `push` and `pull` classes.

Ordering (Push - Pull)

Often, the vertical stacking of columns is desired on smaller screen devices

(tablets/phones) where horizontal space is limited. A typical 2-column page layout is where we can see the most common use case for column ordering. Using the Bootstrap `col-*-push-*` and `col-*-pull-*` CSS classes we're able to utilize the column ordering feature of the Bootstrap grid. On small (mobile) screens we want to show our main content at the top and then the sidebar nav below. This also helps SEO as the main content is closer to the top of the page when Google bot indexes the page.

```
<div class="row">
  <div class="col-sm-9 col-sm-push-3"> main content </div>
  <div class="col-sm-3 col-sm-pull-9"> sidebar </div>
</div>
```

1 - Smaller Layout

main content

sidebar

2 - Larger Layout

sidebar (pulled)

main content (pushed)

An important point to remember is that *both* nesting and push/pull classes can be used to **change column order**. It's also important to think "mobile-first". First create the markup for the desired small screen layout, and then adjust the markup accordingly for the desired large screen layout.

[Read More: A Guide to Column Ordering in Bootstrap](#)

So as you can see the Bootstrap grid is a powerful, well-considered "machine" of `containers`, `columns` and `rows` that enables us to build amazingly flexible responsive layouts. I look forward to the [upcoming Bootstrap 4](#), and I will be revisiting these concepts with the new Bootstrap 4 grid which will introduce more breakpoints, CSS3 flexbox layouts and the long-awaited "equal height rows".



Share this post

Google +

[5 Comments](#) [Codeply](#) [Disqus' Privacy Policy](#)[1 Login](#) ▾[♥ Recommend](#) 6[🐦 Tweet](#)[f Share](#)[Sort by Best](#) ▾

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)**Charissa Santos** • 4 years ago

This was so clearly written and helpful, thank you!

3 ^ | ▾ • Reply • Share >

ruban murthy • 8 months ago

good article

^ | ▾ • Reply • Share >

pganja • 3 years ago

Thank you for a well written explanation.

^ | ▾ • Reply • Share >

Cathy Norris • 3 years ago

excellent article - thank you!

^ | ▾ • Reply • Share >

Becky • 4 years ago

Awesome write-up. Thanks so much!

^ | ▾ • Reply • Share >

[✉ Subscribe](#) [D Add Disqus to your site](#) [Add DisqusAdd](#) [⚠ Do Not Sell My Data](#)



CODEPLY

The Responsive Design Editor

Follow

