We are interested in the requests-per-second metric. The number means that the website can process 91.24 requests per second if there are five requests at a time.

**Note**

Note that debuging is not turned off since we are interested in changes to the session handling speed.

2. Now add the following to the /config/web. php components section:

'session' => array(

'class' => 'yii\web\CacheSession',

'cache' => 'sessionCache',

),

'sessionCache' => array(

'class' => 'yii\caching\MemCache',

),

3. Run ab again with the same settings. This time, you should get better results. In my case, it was 139.07 requests per second. This means Memcache, as a session handler, performed 52% better than the default file-based session handler.

**Note**

Don’t rely on the exact results provided here. It all depends on software versions, settings, and hardware used. Always try to run all tests yourself in an environment where you are going to deploy your application.

4. You can get a significant performance gain by choosing the right session handling backend. Yii supports more caching backends out-of-the-box, including WinCache, XCache, and Zend data cache, which comes with the Zend Server. Moreover, you can implement your own cache backend to use fast noSQL storage, such as Redis.

How it works...

By default, Yii uses native PHP sessions; this means that the filesystem is used in most cases. A filesystem cannot deal with high concurrency efficiently.

Memcache or other platforms perform fine in the following situation:

'session' => array(

'class' => 'yii\web\CacheSession',

'cache' => 'sessionCache',

),

'sessionCache' => array(

'class' => 'yii\caching\MemCache',

),

In the preceding config section, we instruct Yii to use CacheSession as a session handler. With this component, we can delegate session handling to the cache component specified in cache. This time we are using MemCache.