A **best practice**is a technique or methodology that, through experience and research, has proven to reliably lead to superior results. In this blog post, I will talk about few best practices that I have learned over the years which might help you in designing scalable web applications.

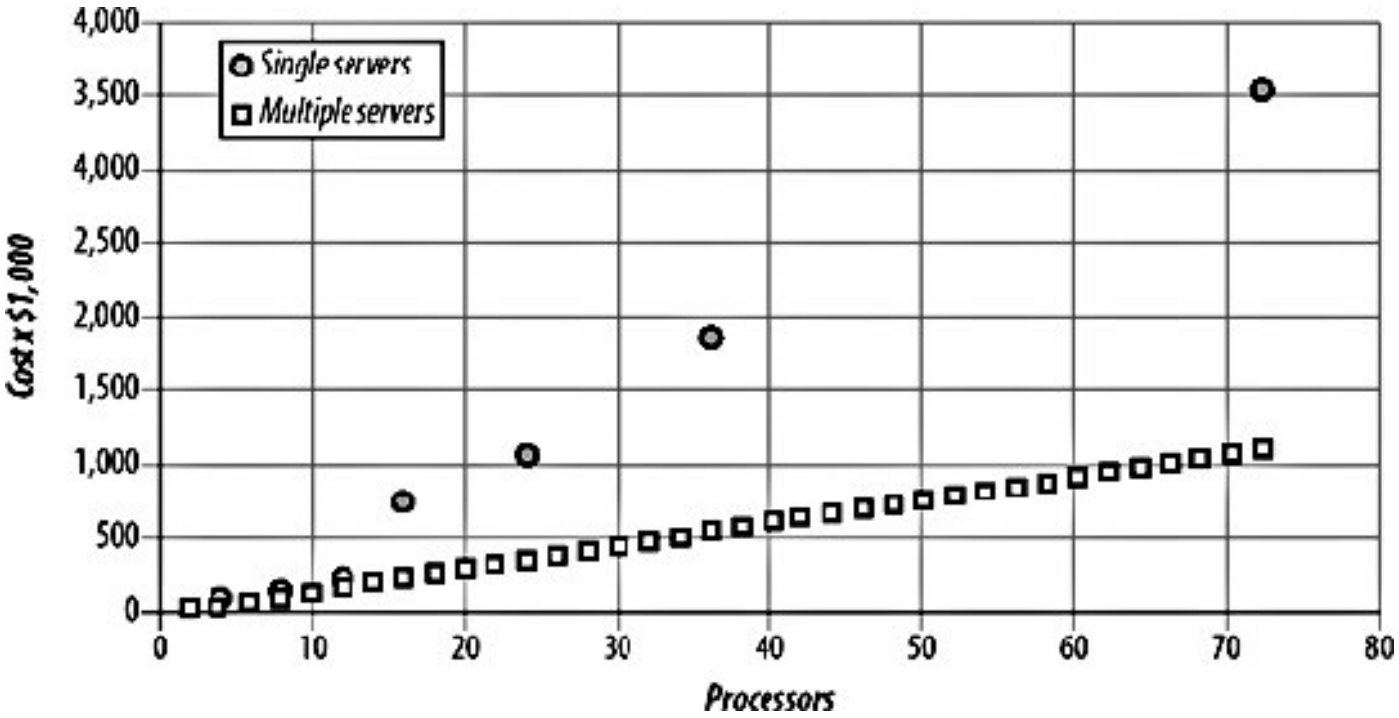
But before we talk about scaling best practices, we should define the meaning of scalability. Most of the people confuse scalability with performance.**Performance** refers to the capability of a system to provide a certain response time. **Scalability** can be defined as ease with which a system or component can be modified to fit the problem area in such a way that the system can accommodate increased usage, increased datasets, and remains maintainable. A system can be performant but might not be scalable. For example, a web application can be very responsive and fast for 100 users and 1GB of data, but it will not be considered scalable if it can't maintain the same speed with 100 times the data with 100 times the users. You achieve scalable architecture when you can handle more load by adding more of the same stuff. You want to be able to scale by throwing money at a problem which means throwing more boxes at a problem as you need them.

Types of Scaling

Vertical scaling : It is about adding more power to the single machine i.e. faster CPU , more RAM , SSD etc . Vertical scalability has a limit and the cost increases exponentially.

Horizontal scaling : It is about handling more requests and load by adding more machines. It requires special attention to application architecture.

Below is the image taken from book "Building Scalable Web Sites by Cal Henderson" which clearly shows that with vertical scaling cost increase exponentially whereas with horizontal scalability cost is linear.



Note : This blog is not about how OpenShift supports horizontal scaling. If you want to learn about that you

can read couple of very good blogs fromSteve on how OpenShift supports AutoScaling and Manual scaling.

Horizontal Scaling Best Practices

Now I will list some of the best practices which will help you scale your web application. Most of these are generic and can be applied to any programming language.

1. Split Your Monolithic Application

The idea behind this practice is to split the monolithic application into groups of functionally related services which can be maintained and scaled together. You can do this via SOA(Service Oriented Architecture),ROA(Resource Oriented Architecture), or by just following good design practices, idea is just to split big monolithic application into smaller applications based on functionality. For example, all user related services can be grouped into one set, search in another, etc. Web scalability is about developing loosely coupled systems. It should be designed in such a way that many independent components communicate with each other. If one component goes down, it should not effect the entire system. This help avoids “single points of failure”. The more decoupled unrelated functionality can be, the more flexibility you will have to scale them independently of one another. As services are now split, the actions we can perform and the code necessary to perform them are split up as well. This means that different teams can become experts in subsets of systems and don’t need to worry about other parts of system. This not only helps in scaling application tier but helps in scaling database tier as well. As rather using single database and going with one choice , you can choose different databases for different needs.

2. Use Distributed Caching

Distributed caching can help in horizontal scalability of a web application by avoiding access to a slow database or filesystem and instead retrieves data directly from the fast local memory. This helps the application in scaling linearly, just by adding more nodes to the cache cluster. A Java web application using a distributed cache can store frequently accessed data such as results of a database query or computation intensive work in a cache. Applications can useMemcached or Infinispan to create distributed cache cluster. Caching is all about minimizing the amount of work a system does. It is advisable that you put caching in its own tier rather than using application servers machines. This will help you in scaling the caching tier independently of application tier.

3. Use CDN

You should use CDN(Content delivery network) to offload traffic from your web application. A content

delivery network or content distribution network (CDN) is a large distributed system of servers deployed in multiple data centers across the Internet. The goal of a CDN is to serve content to end-users with high availability and high performance. CDNs are mostly used for delivering static content like css , images , javascript, static html pages near to the user location. It will find the best possible server which can fulfill the request in the least amount of time by fewer network hops, highest availability, or fewer request. Your application can leverage either Akamai or Amazon CloudFrond for CDN capabilities.

4. Deploy Shared Services To Their Own Cluster

Some applications use file systems to save files uploaded by users, or to store configuration files. You need to replicate these files to all the nodes so that all nodes can use them. With more nodes added, copying files among server instances will occupy all the network bandwidth and consuming considerable CPU resources. To work in a cluster, the solution is to use the database in place of external files, or SAN or use Amazon S3. This will help achieve better scalability.

5. Go Async

The next best practice to scaling is the use of asynchronous calls. If two components X and Y call each other synchronously, then X and Y are tightly coupled, and then either both of them will scale or none of X and Y will scale. This is a characteristic of tightly coupled systems – to scale X, you must scale Y as well and vice versa. For example, it is very common that after user registration email is sent to the user for verification. Now if you tightly couple the user registration service and email service together than scalability of user registration service will be dependent on tje email service. But if you do it asyncronously either through a queue, multicast messaging, or some other means, then you can continue registering users, untill you are sure that the verification email will be sent to user. Synchronous calls stop the entire program execution waiting for a response, which ties all services together leading to cascading failures. This not only impacts scalability but availability of the system as well. In other words, if Y is down then X is down. With async design, X and Y now have independent availability characteristics – X can continue to move forward even if Y is down.

6. Parallelize The Task

There are times when you can divide a single threaded task to multiple smaller tasks which can be run in parallel not only on a single machine but on a cluster of machine. A single thread of tasks will be the scalability bottleneck of the system. Java 7 introduced fork/join framework that helps you take advantage of multiple processors. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application.

7. ## Don’t Store State in the Application Tier

   The golden rule to achieve scalability is not storing state in the application tier but storing state in the database so that each node in the cluster can access the state. Then you can use a standard load-balancer to route incoming traffic. Because all application servers are equal and does not have any transactional state, any of them will be able to process the request. If we need more processing power, we simply add more application servers.

8. ## Use Non-Blocking IO

   The java.nio package allows developers to achieve greater performance in data processing and offers better scalability. The non-blocking I/O operations provided by NIO and NIO.2 boosts Java application performance by getting “closer to the metal” of a Java program, meaning that the NIO and NIO.2 APIs expose lower-level-system operating-system (OS) entry points. In a web application, traditional blocking I/O will use a dedicated working thread for every incoming request. The assigned thread will be responsible for the whole life cycle of the request – reading the request data from the network, decoding the parameters, computing or calling other business logical functions, encoding the result, and sending it out to the requester. Then this thread will return to the thread pool and be reused by other requests. With NIO, multiple HTTP connections can be handled by a single thread and the limit is dependent on amount of heap memory available. You can turn on NIO in Tomcat by changing the protocol attribute of <Connector> element as shown below

```
<Connector
    protocol="org.apache.coyote.http11.Http11NioProtocol"
    port="80"
    redirectPort="8443"
    connectionTimeout="20000"
    compression="on" />
```

References

1. http://www.allthingsdistributed.com/2006/03/a_word_on_scalability.html
2. http://searchsoftwarequality.techtarget.com/definition/best-practice
3. http://www.amazon.co.uk/Building-Scalable-Web-Sites-Henderson/dp/0596102356
4. http://www.infoq.com/articles/ebay-scalability-best-practices
5. http://www.amazon.com/Scalability-Rules-Principles-Scaling-Sites/dp/0321753887
6. Image source http://www.flickr.com/photos/amanda47/435068244/