**Apache load test:**

* First install Kubernetes Metrics Server form <https://artifacthub.io/packages/helm/metrics-server/metrics-server>
* Then deploy the components.yaml file using kubectl apply -f components.yaml command line
* Once the above command executes successfully, run the following command to verify that Metrics Server was installed:  
    
  kubectl get deployment metrics-server -n kube-system
* **Locally.** If the Kubernetes cluster runs on Minikube, a tool for [local Kubernetes deployment](https://www.techtarget.com/searchitoperations/answer/Evaluate-3-ways-to-run-Kubernetes-locally), enable a Metrics Server add-on:

minikube addons enable metrics-server

**Create an application**

Run the following command to install a simple PHP web application in the Kubernetes cluster:

kubectl apply -f https://k8s.io/examples/application/php-apache.yaml

Then, verify the pods were created:

kubectl get pods

* To create an [autoscaling](https://www.techtarget.com/searchcloudcomputing/definition/autoscaling) deployment for CPU, use the following command:

kubectl autoscale deployment php-apache --cpu-percent=80 --min=1 --max=4

This will increase pods to a maximum of four replicas (--max=4) when the PHP web application deployment experiences more than 80% CPU use (--cpu-percent=80) over a sustained period.

* To check the status of Horizontal Pod Autoscaler, run the get command, which displays the current and target CPU consumption. Initially, you might observe an unknown value in the current state, as it takes some time to pull metrics from the Metrics Server and generate the percentage use.

kubectl get hpa

* For a detailed status of the Horizontal Pod Autoscaler, use the describe command to find details such as metrics, events and conditions.

kubectl describe hpa

* Now deploy the infinite-calls.yaml file using kubectl apply -f infinite-calls.yaml
* To deploy this YAML file, use the [Kubectl apply command](https://www.theserverside.com/answer/Kubectl-apply-vs-create-Whats-the-difference) with a path to the file, as seen in Figure 2. A new container has been created to run it.

Kubectl apply -f <path to yaml>

* Once the container is up and running, we can run a shell using /bin/sh interactively in this container. Use the shell to verify that there is a process running to perform web requests to the PHP application infinitely. These never-ending calls introduce load on the application, which in turn consumes the processor resource for hosting the container.

kubectl exec -it <container name>  sh

* You can see the deployed pods, as well as current and target CPU use with the get command and -w flag: kubectl get hpa -w
* Now, forward the service ports to the local host on port 80.

kubectl get services

kubectl.exe port-forward svc/php-apache 80:80

This load test shows the Horizontal Pod Autoscaler kept the application available under increased use conditions. The application still works, as we can see the "OK!" status message in the browser when accessing the PHP application at <http://localhost>

**Reference** <https://www.techtarget.com/searchitoperations/tutorial/Kubernetes-performance-testing-tutorial-Load-test-a-cluster>