Bringing the app to production …

1. **aspects** to make the application provided ready for production:

**Scaling**

Tornado runs a single thread application, and this could limit the CPU cores/thread usage of the hosting server. This is due to the Python GIL (Global Interpreter Lock).

Possible approaches are [[ref.](https://www.tornadoweb.org/en/stable/guide/running.html)]:

- deploying multiple instances of Tornado exploiting multiple CPU/threads (typically it is best to run one process per CPU). Adding ***reuse\_port=true*** to the ***listen()*** calls and simply running multiple copies of the application would achieve this.

This approach runs all the process with the same shared port. It brings three more issues with it: *first*, each child process will have its own IOLoop – so it is important that nothing touches the global IOLoop instance before the fork. *Second*, it is difficult to do zero-downtime updates in this model. *Third,* individual monitoring is more difficult since all the processes share the same port.

- using application load balancers (HAProxy or nginx) and multiple tornado instances ~~(as a process, containers/pod)~~, running on different ports. Outside visitors will access through a single address to the load balancer. To configure this setup it is possible to use the “process groups” feature of ***supervisord*** [[ref.](http://www.supervisord.org/introduction.html)].

- deploying multiple containers, and using a load balancer in front of them, but yet a DB shared-access logic should be implemented (e.g. to avoid possible double mongodb “\_id” a shared list can be used by all the instances).

- deploying Tornado on Kubernetes (e.g. using pod replicas) and a load balancer in front of them [[ref.](https://medium.com/swlh/high-scale-web-server-with-kubernetes-1b35da281dd7)].

**Caching**

Aggressive caching policies [[ref.](https://www.tornadoweb.org/en/stable/guide/running.html" \l "static-files-and-aggressive-file-caching)] providing static files/resources. This can be done specifying ***static\_path*** setting in the application’s *setting,* and using StaticFileHandler.

Browser’s side, caching static resources prevents then sending unnecessary If-Modified-Since or Etag requests. *Static content versioning* is provided out of the box by Tornado, using *static\_url* method in the templates rather than typing the URL of the static file directory in the HTML. Hashing the file content Tornado server sends to the user’s browser headers that will be cached indefinetely. When the file on the server side is updated, the hash will change and the user’s browser will automatically fetch the new file.

***Static\_url()*** can also be configured in file server like nginx.

**Monitoring**

Single process monitoring could be difficult and expensive in case the instances number is high. As mentioned before, if multiple instances run on the same port, independent monitoring would also be hard. As a possible solution, running instances on different ports would enable independent instance monitoring.

On the other hand, for a high level monitoring, a Load balancer monitoring could be a viable solution.

Monitoring a Kubernetes cluster could be the easiest way to go (e.g. deploying prometheus with helm chart using a daemonset).

Another important consideration regarding monitoring: if more tornado instances run on the same port, understanding

**High availability**

Deploying more instances on different locations could help – not sure if with Azure this is possible across Italy (it looks like that we will have [one region – in lombardia](https://news.microsoft.com/europe/2023/06/05/microsoft-announces-its-first-cloud-region-in-italy-accelerating-innovation-and-economic-opportunity/))

2. **actions** to make the application provided ready for production:

**Scaling**

Running Tornado in a ***Kubernetes cluster***, using an Application Load Balancer (ALB) in front of the Tornado deployments, e.g. NGINX, HAProxy, Traefik or Istio. These are the so called **Ingress controllers** (or External load balancer)and they operate at Layer 7. For example Azure has its own Application Gateway. This kind of load balancer exposes the application to external users or services outside the cluster and they require an Ingress controller to be installed (as additional step to the K8s configuration).

Kubernetes makes also use of Internal Load Balancer, which operates at layer 4, to route traffic within the cluster (Azure’s one is azure-load-balancer-internal) [refs.[1](https://spacelift.io/blog/kubernetes-load-balancer), [2](https://spacelift.io/blog/kubernetes-ingress)].

**Monitoring**

Deploying a monitoring system, e.g. Grafana (open-source) or Dynatrace (commercial).

Better would it be on a Kubernetes cluster – but also feasible on a single VM instance, or on a single process.

**High availability**

It is coupled with the scaling approach, caring about the VMs/Nodes locations – interesting for an international deployments, less interesting for a national scale one (since there are not many Azure datacenters in Italy, unless we want to proceed with an hybrid cloud/on-prem approach).

3. **additional developments** to make the application provided ready for production:

The application provides substantially a single endpoint that covers the 5 required actions, namely getting the whole list of student, or a single student by using its id, create a new user, modify or delete and existing one.

The single endpoint makes use of the 4 method involved (GET, POST, PUT, DELETE).

It can be extended to multiple endpoints following the schema:

- GET “/list/” : provide the full students list

- GET “/getStudent/\_id\_student”: provide a student by its id

- POST “/addStudent/”: add a new student using the body of the request as an input,

- PUT “/modStudent/”: modify a student using the body of the request as an input,

- DELETE “/delStudent/\_id\_student”: delete a student using its id.

Improving tests:

- using the ***tornado.testing*** framework more tests can be developed.

- ***Jmeter*** can be used to stress the application to check if there is a breaking point, e.g., creating new users (e.g. the MongoDB ~~collection~~ document max.16mb size, or max Tornado open connections – maybe tens of thousands?), it looks like Jmeter is highly suitable for running parallel http requests.

- ***Postman*** semplifies the operation of creating, and modifying users since it sets Header of the requests automatically – further tests can be created.