Tornado - Practice Sheet

10 Q&A on IOLoop, async handlers, WebSockets, and testing

Q1. What is Tornado and what makes it suited for high-concurrency use cases?

A: Tornado is a Python web framework and async networking library with a non-blocking I/O model. A single-threaded IOLoop handles thousands of concurrent connections efficiently, avoiding the thread-per-request overhead of traditional WSGI servers.

Q2. How does the Tornado IOLoop work with modern Python asyncio?

A: Since Tornado 5, the IOLoop is built on top of asyncio's event loop. tornado.ioloop.IOLoop.current() returns the current loop. You can mix Tornado coroutines with native asyncio code. asyncio.run() starts the loop for both frameworks simultaneously.

Q3. How do you define a request handler in Tornado?

A: Subclass tornado.web.RequestHandler and override HTTP method coroutines: async def get(self), async def post(self). Use self.get_argument(), self.request.body, and self.write() to read input and write output. Map it to a URL in the Application URL spec.

Q4. What is the difference between @tornado.gen.coroutine and async/await?

A: @gen.coroutine with yield was Tornado's pre-3.5 async style and still works. Native async def / await is preferred in Python 3.5+ and is more readable. Under the hood both produce the same IOLoop-driven coroutine, so they interoperate freely.

Q5. How does Tornado's WebSocket handler work?

A: Subclass WebSocketHandler. Override open() (connection established), on_message(message) (data received), and on_close() (connection closed). Call self.write_message(data) to send to the client. Map the handler to a URL like any RequestHandler.

Q6. How does Tornado's template engine handle auto-escaping?

A: Tornado templates auto-escape HTML output by default, preventing XSS. Use {% raw expr %} to render unescaped content. Unlike Jinja2, Tornado templates are compiled to Python code, making them fast but with a smaller feature set.

Q7. How do you implement user authentication in Tornado?

A: Override get_current_user() to return the authenticated user (e.g., decoded from a signed cookie with self.get_secure_cookie). Decorate handlers with @tornado.web.authenticated, which redirects unauthenticated users to login_url.

Q8. How do you enable SSL/TLS in Tornado's HTTPServer?

A: Pass ssl_options={'certfile': '/path/cert.pem', 'keyfile': '/path/key.pem'} to HTTPServer(app, ssl_options=...). HTTPServer can also accept an ssl.SSLContext object for advanced configuration like client certificate verification.

Q9. How do you run Tornado across multiple CPU cores?

A: Call tornado.process.fork_processes(0) to fork one worker per CPU core. Each process runs its own IOLoop. Alternatively, start multiple Tornado processes behind a load balancer. Tornado's non-blocking model means fewer processes are needed than with WSGI.

Q10. How do you write async tests in Tornado?

A: Subclass AsyncHTTPTestCase and override get_app() to return your Application. Use self.fetch('/path') which blocks until the response arrives. @gen_test(timeout=5) decorates test methods that are coroutines, running them on the IOLoop.