Modern C++ for High Performance Web Applications

C++ is a popular systems programming language. While it is not usually considered for web application development, Modern C++ (C++17/20/23) has a compelling set of features and libraries that make it suitable for demanding web workloads.

Why Choose C++ for Web Applications?

- **Predictable Performance:** C++ provides deterministic memory management without a garbage collector, resulting in consistent latency and throughput.
- **Speed:** Native compilation and low-level optimizations allow C++ applications to outperform many managed language counterparts.
- **Resource Efficiency:** Fine-grained control over memory and CPU usage enables efficient scaling, especially in resource-constrained environments or cloud deployments. Topping the chart in cold start, memory usage in AWS Lambda (See Lambda Performance Benchmarks).
- **Low Latency:** Minimal runtime overhead means responses can be extremely fast, making C++ ideal for latency-sensitive APIs and real-time systems.
- **Access to Powerful Libraries:** Leverage a vast ecosystem of high-performance libraries for networking, memory management, task flows, data structures, cryptography, serialization, and more.
- **Hardware Utilization:** Modern C++ supports SIMD, AVX, and other hardware features, allowing you to maximize performance on specific server architectures (e.g., Highway).
- **Performance-Aware Programming:** C++ encourages understanding and optimizing for performance at every level.
- **Direct Code Reuse:** Use robust, mature libraries directly, often without the need for wrappers or bindings.

C++ is well-known for powering databases, browsers, game engines, and system components. Today, you can use it to build everything from raw TCP servers to full-featured REST APIs and microservices, choosing your preferred level of abstraction.

Common Concerns

- **Safety:** Risks of memory corruption, undefined behavior, and crashes are higher compared to managed languages.
- **Security:** Vulnerabilities such as buffer overflows and exploits require careful mitigation.
- **Complexity:** The language has a steep learning curve and requires disciplined engineering practices. It is harder to get right.

Mitigation Strategies:

- Adopt modern C++ best practices (RAII, smart pointers, static analysis).
- Validate all input data rigorously.
- Design services to be restartable and resilient.
- Use sanitizers, memory sentries and fuzzing tools during development.

Notable C++ Web Frameworks and Libraries

Framework	Speed	Features	Production Ready (2025)
Drogon	🚀🚀🚀	★★★	☑
Lithium	🚀🚀🚀	★★½ (no WebSocket?)	☑ (Windows build issues)
Oatpp	🚀🚀	★★★	☑
CrowCpp	🚀🚀	★★★	☑
userver	🚀🚀	★★★	☑
mongoose	🚀🚀🚀	★★★	☑ (great for embedded)
ffread	🚀🚀🚀	★★★	☑
glaze (networking)	?	★★★	?
vixcpp	🚀🚀	★★★	?

Popular Choices:

- Drogon: High-performance, full-featured.
- Lithium: High-performance, easy to use. Linux for now.
- Oatpp: Modern, easy-to-use, and well-documented.
- CrowCpp: Inspired by Python Flask.
- userver: Coroutine-based, feature-rich.
- mongoose: Lightweight, embeddable.
- ffead-cpp: High performance, extensive features.
- glaze: Modern serialization and networking.
- Metaspex: Enterprise-ready backend development platform.
- vixcpp: P2P and high-performance applications. Local-first execution, Asio-powered async I/O, modular architecture..

See TechEmpower benchmarks for performance comparisons.

Choosing Libraries: What to Consider

- **Community & Adoption:** Popular, well-maintained libraries are safer bets.
- **Performance:** Benchmarks and real-world usage. Always benchmark for your use case.
- **Security:** Active patching and secure defaults.
- **Features:** Does it support your use case (REST, WebSockets, GraphQL, etc.)?
- **Ease of Use:** Good documentation and examples.
- **C++ Standard Support:** Prefer libraries using modern C++ idioms(can help with safety, low level APIs may be more performant though, mix and benchmark).
- **License:** Ensure compatibility with your project.

Package Management

- vcpkg
- conan
- CMake's `FetchContent` - see this guide
- Git submodules
- Direct copy in repo - Good for extremely stable headers and utilities. Popularly used with single-header libraries.

Tip: Lock dependencies to specific versions or commits to avoid supply chain risks. Test thoroughly when upgrading.

Build Methods (System?)

- `CMake` - Most widely used.
- Meson
- `Bazel`
- GN - Popular in Google projects.
- Makefiles
- autotools - i.e. Autoconf to create a configure script, Automake to generate Makefile files, and Libtool to assist in building portable libraries. Consumers use Autotools to compile software by running the configure script and then the make command.

Common Libraries for Web Services

- **JSON:** nlohmann/json, RapidJSON, Glaze
- **Cryptography:** argon2, libsodium, jwt-cpp
- **Networking:** Boost.Asio, ZeroMQ
- **Database Drivers and ORMs:** libpqxx (Postgres), mongo-cxx-driver or the driver embedded in your web framework. TinyORM.
- **Validation:** Validation can be handled through custom parsers, so you can use parser combinator libraries like Boost.Spirit.X3 now replaced by Boost.parser and Lexy.
- **Utilities and Standard Library:** Boost, unordered_dense, Folly, Poco, Abseil, EASTL
- **Messaging Queues and In-memory storage** redis-plus-plus, hiredis. These Redis compatible APIs alow you to use most popular in-memory stores like Pogocache, DragonFly and Valkey. rabbitmq-c, AMQP, cppkafka, librdkafka, modern-cpp-kafka
- ML, Math, Matrices - Eigen, mlpack, armadillo
- Identifiers - boost-uuid
- Vector search - usearch

Key Concepts

- **Authentication & Cryptography:** JWT, hashing, signing.
- **Validation:** Input validation, email/phone number parsing (see RFC-5322 for email).
- **Sockets & IPC:** ZeroMQ, Cap'n Proto, gRPC.
- **Messaging Queues:** Kafka, RabbitMQ.

- **Cloud SDKs:** AWS, Azure, GCP (Google Cloud C++, AWS C++ SDK, AWS Lambda C++ Runtime, Azure C++ SDK).
- **Database Clients:** MongoDB, Postgres, Cassandra, CouchDB, ScyllaDB
- **Data Structures:** High-performance hashmaps, sets - ankerl's unordered_dense.h. See these: ankerl's hashmap benchmark, tessil's, attractivechaos's, jacksonallan's
- **GraphQL:** Emerging support in some frameworks.
- **DevOps:** Modern C++ DevOps

Good Practices

- Use modern C++ features (smart pointers, `std::optional`, `std::expected`).
- Reserve exceptions for truly exceptional cases.
- Prefer prepared statements or constant strings for database queries to avoid SQL injection.
- Write comprehensive tests and use sanitizers (ASan, UBSan).
- Monitor for deadlocks, especially when using coroutines or async code.
- Study the documentation and source code of dependencies. The source code is the single source of truth. Documentation can be outdated. AI can be outdated or just wrong.
- When confused, benchmark, monitor for performance regressions, profile hot paths, benchmark different approaches: Google Benchmark is a nice microbenchmarking lib. Compiler explorer is a good tool to quickly test patterns and compare compiler output. Other useful tools are Quickbench (Quick microbenchmarking), Cpp Insights (See your source code with the eyes of a compiler).
- Spend time to setup your dev environment, LSPs, debuggers are really useful. Lots of time is spent debugging.

Strategies for Using C++ in Web Backends

1. C++ for Latency-Critical Microservices

Use C++ to implement microservices that are performance bottlenecks or require deterministic latency—such as authentication, inferencing, video/image processing, real-time analytics, search engines, recommendation engines, or protocol gateways. These services benefit from C++'s predictable memory management, extensive library support,direct hardware access, and minimal runtime overhead.

- **Example:** Meta (Facebook) uses C++ (and C) for core messaging and real-time infrastructure.
- **Integration:** Expose C++ services via REST/gRPC, and orchestrate them with higher-level languages (Go, Node.js, Python) or a more general C++ service for non-critical paths.
- **Benefits:** Achieve sub-millisecond response times, reduce cloud costs, and maximize hardware utilization.

2. Full-Stack C++ Web Applications

Build entire web applications or APIs in C++ using modern frameworks (Drogon, Oatpp, userver). This approach is ideal for teams with strong C++ expertise or applications where performance, efficiency, and control are paramount.

- **Advantages:**

    - End-to-end type safety and performance.
    - Unified toolchain and deployment.
    - Fine-grained control over threading, memory, and networking.
- **Use Cases:** Financial trading platforms, IoT backends, embedded web servers, SaaS with strict SLAs.

3. Hybrid Approach: C++ with Scripting/Managed Languages

Combine C++ for core logic with scripting or managed languages for flexibility and rapid development.

- **Pattern:** Core libraries, performance-sensitive modules, or plugins in C++; business logic, orchestration, and UI in Python, C# and JS (with HTML and CSS).
- **Interoperability:** Use FFI (Foreign Function Interface), REST/gRPC, or WebAssembly to bridge languages.
- **Example:** Use C++ for a high-throughput data processing engine, and expose it to a Go, C# or Node.js Express API.

4. C++ for Edge and Serverless

Deploy C++ in serverless (AWS Lambda, Azure Functions) or edge environments where cold start time, memory footprint, and execution speed are critical.

- **Benefits:** C++ consistently tops benchmarks for cold start and memory usage (see Lambda Performance Benchmarks).
- **Use Cases:** Real-time APIs, IoT gateways, CDN edge logic.

5. C++ for Specialized Backend Components

Leverage C++ for components like:

- **Custom protocol servers** (e.g., WebSocket, MQTT, custom TCP/UDP).
- **High-performance caches** and in-memory stores.
- **Data serialization/deserialization** (e.g., JSON, zpp::bits, Protobuf, Cap'n Proto).
- **Cryptography and security modules**.
- **Media Servers**

Summary

C++ is not just for systems programming—it is a first-class choice for high-performance web backends. Whether you use it for critical microservices, full-stack APIs, or specialized components, C++ delivers unmatched speed, efficiency, and control for demanding web workloads.

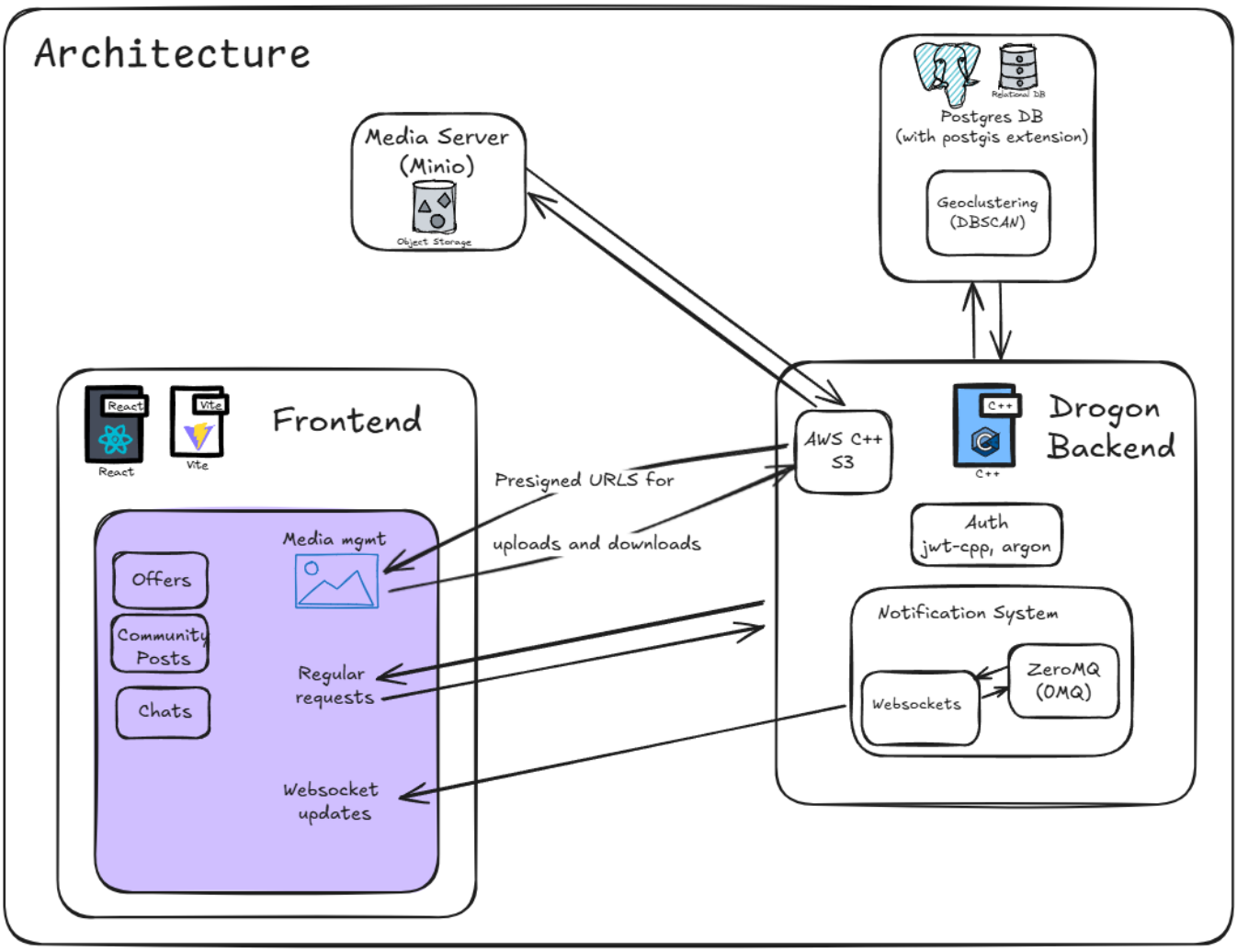
Further Reading & Research

- AWS Lambda Concurrency
- Concurrency Evaluation in C++
- C++ with AWS Lambda Custom Runtime
- Learn CMake

- awesome-cpp
- Reddit discussion on useful tools

Modern C++ empowers you to build web applications and services that are fast, efficient, and robust—provided you follow best practices and leverage the right tools.

Examples and Demo's



Drogon C++ backend for simple marketplace app. Demos Auth, PubSub and Websockets.

Drogon + HTMX

Drogon used in a Discord Bot by Ruslan