



## OVERVIEW & SETUP

IBM WebSphere Enterprise technology

```
# Install
npm install ibm-websphere
# Or: pip install / gem install / go get
```

Check official documentation at [docs.ibmwebsphere.com](https://docs.ibmwebsphere.com)

Verify installation before proceeding

## CORE CONCEPTS

Key abstraction: understand the primary model

Configuration: define behavior through config/code

Integration: connects with existing systems

```
# Initialize
const client = new IBMWebSphere({ /*
config */ })
```

## QUICK START

```
# Minimal working example
const app = new IBMWebSphere()
app.configure({ ... })
await app.start()
console.log('IBM WebSphere is running')
```

See official quickstart guide for language-specific setup

## CONFIGURATION

```
# config.yaml or config.json
ibm:
  host: localhost
  port: 8080
  timeout: 30s
```

Use environment variables for secrets

```
export IBM_WEBSPHERE_SECRET=my-secret
```

## BASIC OPERATIONS

```
# Create / write
client.create({ name: 'example' })
# Read / query
const result = await client.find({
  filter: {} })
# Update
await client.update(id, { field:
newValue })
# Delete
await client.delete(id)
```

## INTEGRATION

Integrates with common frameworks and tools

```
# Express.js integration example
app.use(IBMWebSphereMiddleware({ config
}))
```

SDK available for: Node.js, Python, Java, Go, .NET

REST API available for language-agnostic integration

Check community-maintained integrations

## SECURITY

Always use authentication and authorization

Encrypt data at rest and in transit (TLS/SSL)

```
# Enable TLS
tls: { cert: ./cert.pem, key: ./key.pem
}
```

Rotate credentials and API keys regularly

Follow security best practices in official docs

## MONITORING

Expose health check endpoint

```
# Health check
GET /health { status: 'ok', uptime:
1234 }
```

Monitor key metrics: latency, throughput, error rate

Set up alerts for anomalies

Log requests with structured logging (JSON format)

## PERFORMANCE TIPS

Use connection pooling for network resources

Enable caching for frequently accessed data

Batch operations to reduce round trips

```
# Async/parallel processing
await Promise.all([op1(), op2(), op3()])
```

Profile before optimizing measure, then improve

## TESTING

```
# Unit test example
describe('IBM WebSphere', () => {
  it('should work', async () => {
    const result = await
client.doSomething()
    expect(result).toBeDefined()
  })
})
```

Use mocks for external dependencies in unit tests

## CLI REFERENCE

|                          |                               |
|--------------------------|-------------------------------|
| <code>ibm --help</code>  | <code># all commands</code>   |
| <code>ibm version</code> | <code># version info</code>   |
| <code>ibm config</code>  | <code># configuration</code>  |
| <code>ibm status</code>  | <code># current status</code> |
| <code>ibm logs</code>    | <code># view logs</code>      |

## COMMON GOTCHAS

Read the changelog before upgrading major versions

Check resource limits (memory, connections, rate limits)

Implement graceful shutdown to avoid data loss

Keep dependencies updated for security patches

Test in staging environment before deploying to production