

Maximizing the number of concurrent browsers

For performance testing, you can increase the number of concurrent Chrome browsers that can run on an AIQ Test Node by optimizing the node's compute and storage resources.

The maximum number of concurrent browsers depends on the application under test, the complexity of the test, browser activity, available CPU and memory, and storage performance.

As a general starting point, an AWS EC2 node using standard EBS storage may support approximately **20 to 30 concurrent Chrome browsers**. Higher concurrency can be achieved by using EC2 instance types that provide local NVMe SSD instance storage and by configuring AIQ to use that storage.

“ **Important:** These numbers are guidelines, not fixed limits. The appropriate browser count must be determined through performance testing with the application under test.

Using NVMe SSD Instance Storage

Storage I/O can become a bottleneck when running a large number of concurrent browsers. Browser processes generate significant disk activity, including cache and temporary-file operations.

AWS EC2 instance types such as the **m6id** family include local NVMe SSD instance storage. For example, current AWS specifications list:

- `m6id.2xlarge`: 8 vCPUs, 32 GiB memory, 474 GB NVMe SSD
- `m6id.4xlarge`: 16 vCPUs, 64 GiB memory, 950 GB NVMe SSD

These NVMe SSD volumes provide substantially higher I/O performance than typical baseline EBS configurations.

Important considerations

NVMe instance storage is **ephemeral**. Data stored on an instance store volume persists only for the lifetime of the associated EC2 instance. It should therefore be treated as temporary working storage rather than permanent storage.

AIQ files, configuration files, or other data that must survive an instance stop or termination should remain on persistent storage.

Because the NVMe storage is not persistent, the node must prepare the storage and copy the required AIQ files to it each time the instance is initialized.

Configuring AIQ to Use NVMe Storage

The recommended setup uses an initialization script, such as `nvme_setup.sh`, to prepare the performance node.

The script should:

1. Detect and mount the available NVMe instance-store device.
2. Copy the GraalVM installation to the NVMe storage.
3. Copy the `aiq_distribution` directory to the NVMe storage.
4. Configure Chrome to use a local cache directory on the NVMe storage.
5. Start AIQ from the NVMe storage.

The Chrome configuration can be added to:

`aiq_distribution/NetworkProxy/config/global.json`

Add the following configuration:

```
{
  "name": "experimentalDS3",
  "value": "--disk-cache-dir=cache|--disk-cache-size=1"
}
```

This configuration redirects the Chrome disk cache to the local storage used by the AIQ node.

Setting Up an AWS EC2 Performance Node

1. Prepare the NVMe setup script

Add the `nvme_setup.sh` script to the AIQ distribution on the EC2 instance.

The script should detect the available NVMe instance-store volume, mount it, and copy the required AIQ and GraalVM directories to the local storage.

The original script assumes that AIQ is installed in:

```
/opt/aiq/aiq_distribution
```

If AIQ is installed in another location, update the script accordingly.

“ **Note:** The exact NVMe device name can vary between EC2 instance types and operating systems. The setup script should detect the device rather than assuming a fixed device name.

2. Configure Chrome disk caching

Modify:

```
aiq_distribution/NetworkProxy/config/global.json
```

Add:

```
{
  "name": "experimentalDS3",
  "value": "--disk-cache-dir=cache|--disk-cache-size=1"
}
```

The cache directory should reside on the NVMe-backed AIQ installation.

3. Start AIQ using systemd

For Linux performance nodes, start AIQ through a `systemd` service rather than `/etc/rc.local`.

Using `systemd` also allows you to increase operating-system resource limits. This is particularly important when running many concurrent browsers because the default per-process file descriptor limit may be too low.

Create:

```
/etc/systemd/system/appvance.service
```

Use the following configuration and adjust the paths and Java version for the environment:

```
[Unit]
Description=Start Appvance IQ services
After=syslog.target

[Service]
Environment=PATH=/opt/graalvm-ce-java11-21.3.3.1/bin:/sbin:/bin:/usr/sbin:/usr/bin
Environment=JAVA_HOME=/opt/graalvm-ce-java11-21.3.3.1
User=ec2-user
Type=simple

LimitNOFILE=10000000
LimitNPROC=20000

ExecStart=/bin/bash /opt/aiq/aiq_distribution/TestNode/nvme_setup.sh
WorkingDirectory=/opt/aiq/aiq_distribution/TestNode

TimeoutStartSec=0

[Install]
WantedBy=multi-user.target
```

“ **Note:** Verify the Java/GraalVM version and installation path before using this configuration. The paths above are examples from the original configuration and may not match the AIQ version installed on your node.

After creating the service, run:

```
sudo systemctl daemon-reload
sudo systemctl enable appvance
sudo systemctl start appvance
```

To verify the service:

```
sudo systemctl status appvance
```

The original article contains a typo (`systemclt enable`). The correct command is:

```
systemctl enable appvance
```

Resource Limits

When increasing browser concurrency, make sure the operating system can support the required number of processes and open files.

The example service uses:

```
LimitNOFILE=10000000
LimitNPROC=20000
```

These values are intentionally high and should be validated against the requirements of the specific environment. Increasing limits does not by itself guarantee higher browser concurrency; CPU, memory, storage I/O, network capacity, and the application under test can all become bottlenecks.

Example Concurrency Results

The original performance testing produced the following results:

EC2 Instance	Concurrent Chrome Browsers
m6id.2xlarge	40
m6id.4xlarge	80

These results were obtained with a Test Designer test case and should be treated as **example performance results rather than guaranteed capacity**.

Actual concurrency can vary depending on:

- Application complexity
- Number and type of requests generated by each browser
- JavaScript execution
- Browser memory consumption
- CPU utilization
- Network traffic
- Test data
- AIQ configuration
- Chrome version
- Operating-system configuration
- Storage performance

For this reason, perform a capacity test before selecting the final number of concurrent browsers for a performance environment.

Amazon Linux Version

The original version of this article specifically targeted **Amazon Linux 2**.

Amazon Linux 2 reached end of life on June 30, 2026. AWS recommends migrating to **Amazon Linux 2023**.

Therefore, new performance nodes should use an AWS-supported operating system and the corresponding AIQ installation and startup configuration.

If an existing environment still uses Amazon Linux 2, treat the configuration in this article as a legacy setup and plan migration to Amazon Linux 2023.

EBS Storage Considerations

EBS performance depends on the volume type and its provisioned performance.

For example, Amazon EBS  volumes provide a baseline of **3,000 IOPS and 125 MiB/s throughput**, with additional IOPS and throughput available through provisioning.

Therefore, the storage bottleneck should not be described simply as a limitation of a "60 GB EBS volume." When investigating concurrency limitations, monitor the actual EBS IOPS, throughput, latency, and utilization of the node.

For workloads that generate substantial browser disk activity, local NVMe instance storage can be an effective way to reduce EBS-related storage contention.

Troubleshooting and Capacity Testing

If increasing the number of browsers causes execution failures or significantly longer response times, monitor the following resources during the test:

1. CPU utilization
2. Memory utilization
3. EBS IOPS and throughput
4. NVMe utilization
5. Disk latency
6. Network utilization
7. Number of open files
8. Number of running processes
9. Chrome memory consumption
10. AIQ/Test Node logs

Increase concurrency gradually rather than immediately configuring the maximum expected browser count.

For example:

20 browsers → 40 browsers → 60 browsers → 80 browsers

At each level, verify that the node remains stable and that response times and error rates remain within the requirements of the performance test.

Last updated: September 2, 2026

If you have any feedback on this topic, please contact us at documentation@appvance.ai.

Revision #5

Created 2026-07-31 15:40:26 UTC by Admin

Updated 2026-09-03 02:43:02 UTC by mmora