Selenium Grid Distributed Configuration Exploration Report

This report explores the impact of configuring Selenium Grid 4.8.3 components separately (Hub, Distributor, Session Map, Event Bus, and Nodes) compared to a standalone setup. The objective is to understand performance, scalability, and architectural implications of distributed configurations, especially for environments with desktop and mobile nodes (e.g., Appium).

1. Selenium Grid Components Overview

Component	Purpose	Default Port(s)
Router (Hub)	Entry point for WebDriver requests; routes to Distributor/Sessions	4444
Distributor	Allocates sessions to available Nodes	5553
Session Map	Tracks active sessions and node ownership	5556
Event Bus	Enables inter-component communication	4442 (Pub), 4443 (Sub)
Node	Executes browser or mobile sessions	Custom (e.g., 5555, 4723)

2. Performance and Scalability Impact

Area	Effect	Explanation
Scalability	Significant Improvement	Each service can be scaled independently to handle high parallel t
Stability	Improved	Crashes in one component don’t affect the others; easier recovery
Resource Utilization	More Efficient	Allows targeted resource allocation (CPU, memory) per componen
Network Overhead	Slightly Higher	More HTTP calls between components introduce minimal latency.
Setup Complexity	Higher	Requires multiple services or containers, monitoring, and coordina
Load Balancing	Better	Distributor efficiently allocates sessions across diverse nodes.
Maintainability	Improved	Individual component updates without full grid downtime.

3. Configuration Recommendations

- Use distributed setup if you run 20+ parallel sessions or multiple node types (desktop + mobile). - Keep Event Bus and Distributor on powerful machines to minimize bottlenecks. - Use Docker Compose or Kubernetes for orchestration. - Maintain consistent port exposure and environment variables across services. - For Appium integration, assign dedicated mobile nodes with limited max-sessions (usually 1 per device).

In conclusion, configuring Selenium Grid components separately leads to substantial gains in scalability and stability for medium to large test environments, especially when managing diverse workloads. The trade-off is a modest increase in setup and maintenance complexity, which can be mitigated using container orchestration tools.