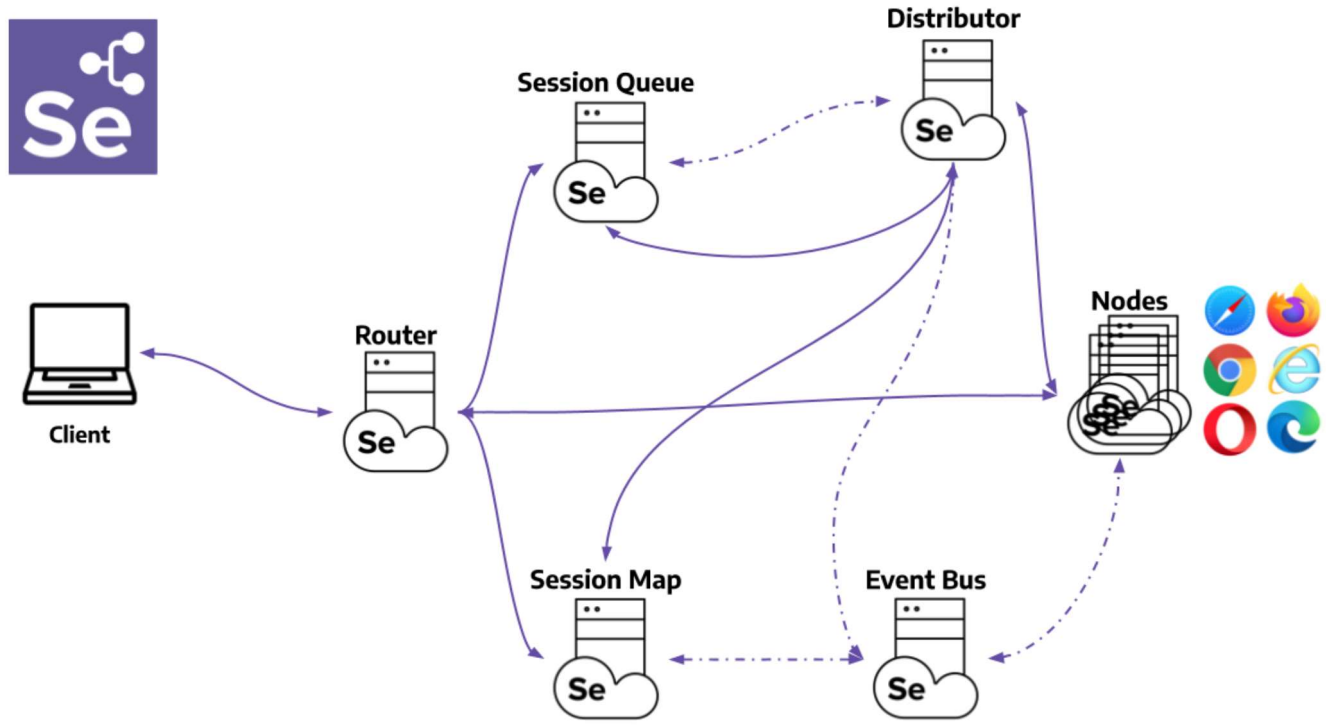
Selenium

WebDriver BiDi is the future of cross-browser automation. Read all about it!

Selenium Grid Components

Understand how to use the different Grid components

Selenium Grid 4 is a ground-up rewrite from previous versions. In addition to a comprehensive set of improvements to performance and standards compliance, the different functions of the grid were broken out to reflect a more modern age of computing and software development. Purpose-build for containerization and cloud-distributed scalability, Selenium Grid 4 is a wholly new solution for the modern era.



Router

The **Router** is the entry point of the Grid, receiving all external requests, and forwards them to the correct component.

If the **Router** receives a new session request, it will be forwarded to the **New Session Queue**.

If the request belongs to an existing session, the **Router** will query the **Session Map** to get the **Node** ID where the session is running, and then the request will be forwarded directly to the **Node**.

The **Router** balances the load in the Grid by sending the requests to the component that is able to handle them better, without overloading any component that is not needed in the process.

Distributor

The **Distributor** has two main responsibilities:

Register and keep track of all Nodes and their capabilities

A **Node** registers to the **Distributor** by sending a **Node** registration event through the **Event Bus**. The **Distributor** reads it, and then tries to reach the **Node** via HTTP to confirm its existence. If the request is successful, the **Distributor** registers the Node and keeps track of all **Nodes** capabilities through the **GridModel**.

Query the New Session Queue and process any pending new session requests

When a new session request is sent to the **Router**, it gets forwarded to the **New Session Queue**, where it will wait in the queue. The **Distributor** will poll the **New Session Queue** for pending new session requests, and then finds a suitable **Node** where the session can be created. After the session has been created, the **Distributor** stores in the **Session Map** the relation between the session id and **Node** where the session is being executed.

Session Map

The **Session Map** is a data store that keeps the relationship between the session id and the **Node** where the session is running. It supports the **Router** in the process of forwarding a request to the **Node**. The **Router** will ask the **Session Map** for the **Node** associated to a session id.

New Session Queue

The **New Session Queue** holds all the new session requests in a FIFO order. It has configurable parameters for setting the request timeout and request retry interval (how often the timeout will be checked).

The **Router** adds the new session request to the **New Session Queue** and waits for the response. The **New Session Queue** regularly checks if any request in the queue has timed out, if so the request is rejected and removed immediately.

The **Distributor** regularly checks if a slot is available. If so, the **Distributor** polls the **New Session Queue** for the first matching request. The **Distributor** then attempts to create a new session.

Once the requested capabilities match the capabilities of any of the free **Node** slots, the **Distributor** attempts to get the available slot. If all the slots are busy, the **Distributor** will send the request back to the queue. If request times out while retrying or adding to the front of the queue, it will be rejected.

After a session is created successfully, the **Distributor** sends the session information to the **New Session Queue**, which then gets sent back to the **Router**, and finally to the client.

Node

A Grid can contain multiple **Nodes**. Each **Node** manages the slots for the available browsers of the machine where it is running.

The **Node** registers itself to the **Distributor** through the **Event Bus**, and its configuration is sent as part of the registration message.

By default, the **Node** auto-registers all browser drivers available on the path of the machine where it runs. It also creates one slot per available CPU for Chromium based browsers and Firefox. For Safari, only one slot is created. Through a specific configuration, it can run sessions in Docker containers or relay commands.

A **Node** only executes the received commands, it does not evaluate, make judgments, or control anything other than the flow of commands and responses. The machines where the **Node** is running does not need to have the same operating system as the other components. For example, A Windows **Node** might have the capability of offering IE Mode on Edge as a browser option, whereas this would not be possible on Linux or Mac, and a Grid can have multiple **Nodes** configured with Windows, Mac, or Linux.

Event Bus

The **Event Bus** serves as a communication path between the **Nodes**, **Distributor**, **New Session Queue**, and **Session Map**. The Grid does most of its internal communication through messages, avoiding expensive HTTP calls. When starting the Grid in its fully distributed mode, the **Event Bus** is the first component that should be started.

Running your own Grid

Looking forward to using all these components and run your own Grid? Head to the "Getting Started" section to understand how to put all these pieces together.

Last modified July 5, 2023: fix display of grid components image [deploy site] (d585f187431)


digital.ai™


applitools
AI-Powered Visual Testing & Monitoring


Perfecto
by Perforce


LAMBDATEST


SUBJECT7
QUALITY IS A CULTURE