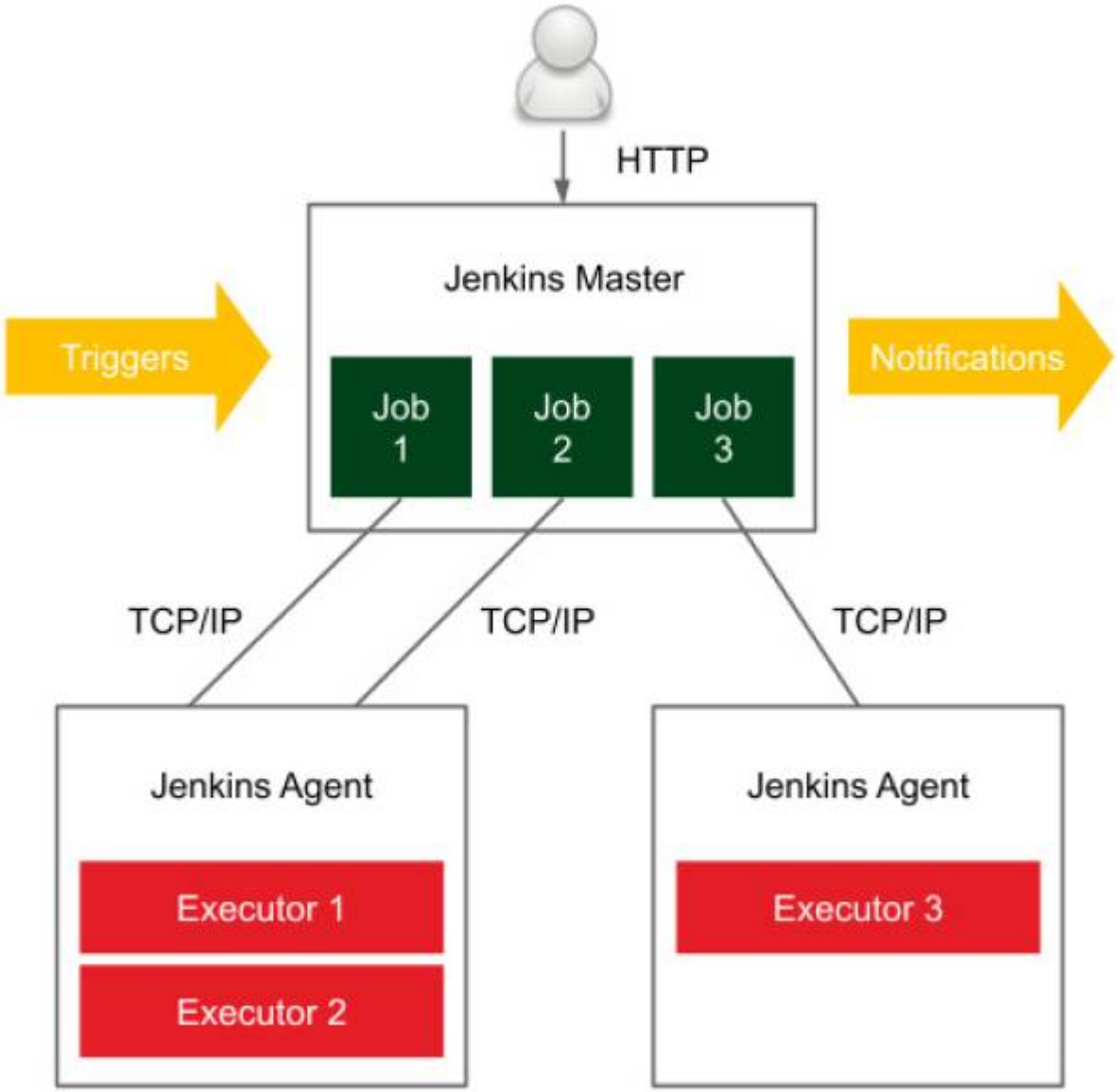
Jenkins architecture

Master and agents

Jenkins server (master) delegates builds and execution tasks to agent (slave) instances.



Jenkins controller (master) is responsible for the following:

- Receiving build triggers (for example, after a commit to GitHub)
- Sending notifications (for example, email or Slack messages sent after a build failure)
- Handling HTTP requests (interaction with clients)
- Managing the build environment (orchestrating the job executions on agents)

The build agent is a machine that takes care of everything that happens after the build has started.

Scalability

When Jenkins instance become overloaded or unresponsive you can scaling it up, either vertical or horizontal.

Vertical scaling

Vertical scaling means that more resources (more RAM, CPU cores, HDD drives etc.) are applied to a machine. For example: having a single Jenkins master set on ultra-efficient hardware is a maintenance advantage. Any upgrades, scripts, security settings, role assignments, or plugin installations have to be done in one place only.

Horizontal scaling

Horizontal scaling means that more master instances are launched with some significant advantages:

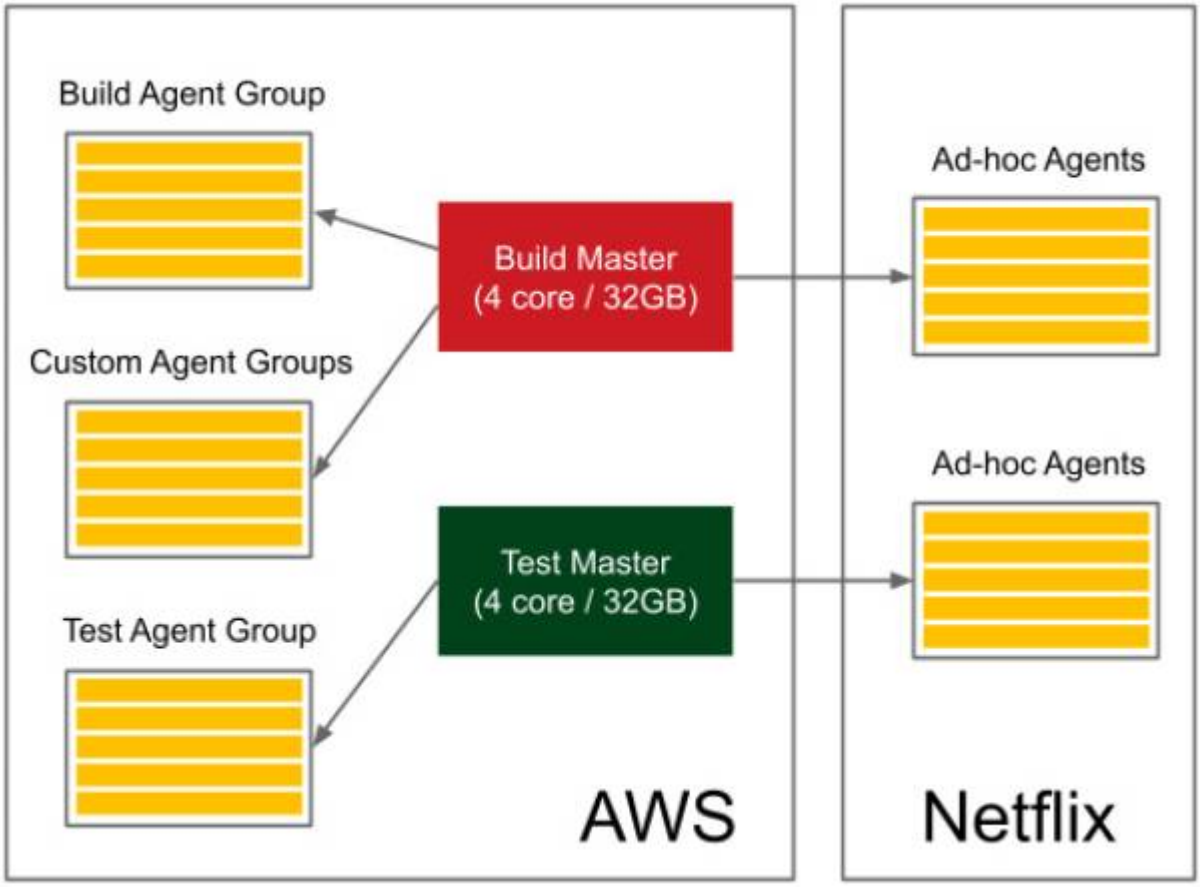
- Master machines don't need to be special, in terms of hardware.
- Different teams can have different Jenkins settings (for example, different sets of plugins).
- Teams usually feel better and work with Jenkins more efficiently if the instance is their own.
- If one master instance is down, it does not impact the whole organization.
- The infrastructure can be segregated into standard and mission-critical.

Test and production instances

How to test the Jenkins upgrades, new plugins, or pipeline definitions? Jenkins is critical to the whole company. It guarantees the quality of the software and, in the case of continuous delivery, deploys to the production servers. That is why it needs to be highly available, and it is definitely not for the purpose of testing. It means there should always be two instances of the same Jenkins infrastructure – test and production.

Sample architecture

We already know that there should be agents and (possibly multiple) masters and that everything should be duplicated in the test and production environments. However, what would the complete picture look like? Let's look at the example of Netflix:



They have test and production master instances, with each of them owning a pool of agents and additional ad hoc agents. Altogether, it serves around 2,000 builds per day. One part of their infrastructure is hosted on AWS and another part is on their own servers.

Setting agents (slave)

Agents always communicate with the Jenkins master using one of the protocols: SSH (sshd) or Java web start. At a higher level, we can attach agents to the master in various ways:

- Static versus dynamic: The simplest option is to add agents permanently in the Jenkins master. The drawback of such a solution is that we always need to manually change something if we need more (or fewer) agent nodes. A better option is to dynamically provision agents as they are needed.
- Specific versus general-purpose: Agents can be specific (for example, different agents for the projects based on Java 8 and Java 11) or general-purpose (an agent acts as a Docker host and a pipeline is built inside a Docker container).

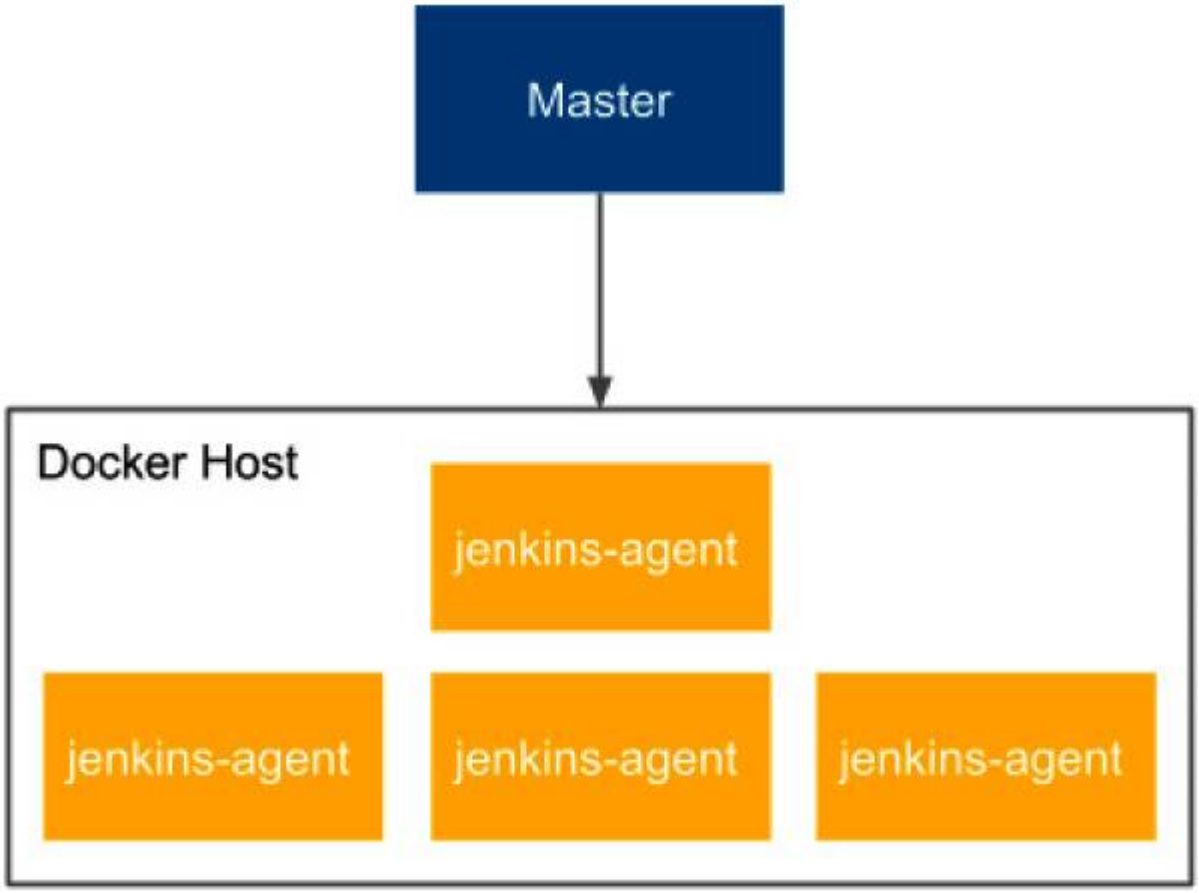
These differences resulted in four common strategies for how agents are configured:

- Permanent agents
- Permanent Docker host agents
- Jenkins Swarm agents
- Dynamically provisioned Docker agents (focus)
- Dynamically provisioned Kubernetes agents

Let's focus on one strategy. Other strategies can be dived in → here.

Dynamically provisioned Docker agents

Another option is to set up Jenkins to dynamically create a new agent each time a build is started. Such a solution is obviously the most flexible one, since the number of agents dynamically adjusts to the number of builds.



Docker agent mechanism used step by step:

1. When the Jenkins job is started, the master runs a new container from the `jenkins/agent` image on the agent Docker host.
2. The `jenkins/agent` container starts the Jenkins agent and attaches it to the Jenkins master's nodes pool.
3. Jenkins executes the pipeline inside the `jenkins/agent` container.

After the build, the master stops and removes the agent container. Information

Running the Jenkins master as a Docker container is independent of running Jenkins agents as Docker containers. It's reasonable to do both, but any of them will work separately.

Custom Jenkins images So far, we have used Jenkins images pulled from the internet. We used jenkins/jenkins for the master container and jenkins/agent (or jenkins/inbound-agent or jenkins/ssh-agent) for the agent container. However, you may want to build your own images to satisfy the specific build environment requirements. In this section, we will cover how to do it.

Custom Jenkins images

Jenkins images can be pulled from the internet

- `jenkins/jenkins` for the master container
- `jenkins/agent` (or jenkins/inbound-agent or jenkins/ssh-agent) for the agent container.

However, you may want to build your own images to satisfy the specific build environment requirements.

Building a custom Jenkins agent

Let's start with the agent image because it's more frequently customized. The build execution is performed on the agent, so it's the agent that needs to have the environment adjusted to the project we want to build – for example, it may require the Python interpreter if our project is written in Python. The same applies to any library, tool, or testing framework, or anything that is needed by the project.

There are four steps to building and using the custom image:

1. Create a Docker file.
2. Build the image.
3. Push the image into a registry.
4. Change the agent configuration on the master.

Building the Jenkins master

Why would we also want to build our own master image? Imagine the following scenario: your organization scales Jenkins horizontally, and each team has its own instance. There is, however, some common configuration – for example, a set of base plugins, backup strategies, or the company logo. Then, repeating the same configuration for each of the teams is a waste of time. So, we can prepare the shared master image and let the teams use it. Jenkins is natively configured using `XML files`, and it provides the `Groovy-based DSL` language to manipulate them. That is why we can add the Groovy script to the Dockerfile in order to manipulate the Jenkins configuration. All possibilities are well described on the → GitHub-Page.

Based on the book: „Continuous Delivery with Docker and Jenkins, 3rd Edition - Third Edition By Leszko“
