Scaling Jenkins on Kubernetes

Jenkins Scalability

When using the **standalone** Jenkins server, you do not have to worry about **multiple servers and nodes**.

However, the issue with this setup is that your server can become overloaded with numerous jobs running at the same time.

There are ways to solve this problem by increasing the number of **executors, but you soon end up hitting a performance limit.**

To overcome this problem, you can offload some of the jobs to different machines called Jenkins's agents (Have java installed as the requirement).

Scalability is a measure that shows the ability of a system to expand its capabilities to handle additional load.

One of the strongest sides of Jenkins is that it has a scaling feature out-of-the-box.

Jenkins's scaling is based on the **controller/agents' model**, where you have several agent instances, and one main Jenkins's instance called the controller that is responsible mainly **for distributing jobs across the agents.**

It is not secure to build your jobs through the Jenkins controller.

Jenkins's agents run a small program (SSH Agent) that communicates with the Jenkins controller to check if there's any job it can run.

When Jenkins finds a job scheduled, it transfers the build to the agent.

Scaling Jenkins on Kubernetes

Taking it a step further, running a container orchestrator such as Kubernetes, you can make Jenkins do smarter things.

The Jenkins controller is responsible for the hosting configuration and the distribution of jobs to multiple agents.

However, you do not need the agents to be **preexisting**, they can be created **on the fly**, as in when they are required.

Advantages of scaling Jenkins on Kubernetes

Auto healing is possible

If your build or your agent gets corrupted, you no longer need to worry — Jenkins will remove the unhealthy instance and spin up a new one.

Run builds in parallel

You no longer must plan the executors and limit them; instead, Jenkins will spin up an agent instance and run your build in it.

Even load distribution

Kubernetes manages loads well, and it will make sure your Jenkins agents spin up in the best available server, which makes your builds faster and more efficient.

In my company we setup dockerfile which is highly recommended to make your continuous integration infrastructure replicable and have it ready for setup from scratch.

```
FROM jenkins/jenkins:lts-slim

# Pipelines with Blue Ocean UI and Kubernetes
RUN jenkins-plugin-cli --plugins blueocean kubernetes
```

Create a dockerfile and run this command in the terminal

docker build .

Push this image to the docker hub or ECR

FROM [image_name]:[tag] - this line means that our dockerfile will be based on an existing image. It is obvious that for our dockerfile, we take the Jenkins base image, which you can find in the official Docker repository.

RUN [command] - this line means that we are going to run this command inside the image during the build process. In our case we use the RUN command to install different common Jenkins plugins, which can be required for your Jenkins master.

You can install additional plugins just by adding similar commands inside your dockerfile like we did above. However, one of the plugins above is critical for installation. It is the Kubernetes plugin (It is used to configure jenkins). This plugin is designed to implement Jenkins's scaling on top of a Kubernetes cluster.

This is how it works in the company.

We going to build the above image and push it to the remote repository. This remote repository can be docker hub or ECR (elastic container registry). This can be automated using Jenkins's (CI/CD) pipeline.

We are going to create Kubernetes deploy manifest and use the image of jenkins created by the CI/CD pipeline to deploy to the cluster.