Questions to Answer

1. What was the problem you were trying to address, and how did you identify/recognize that it existed?

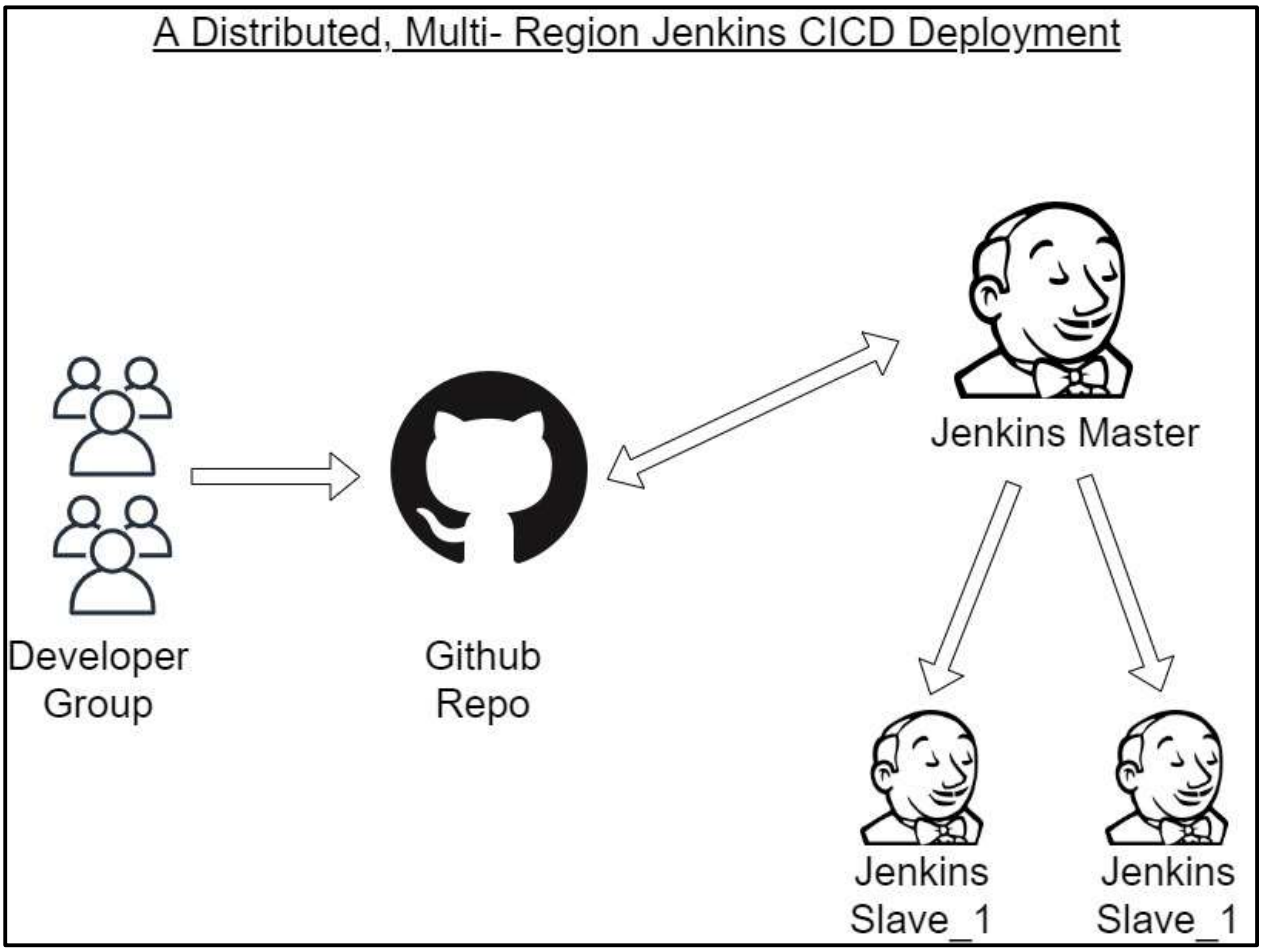

Figure 1: A Distributed Jenkins CI/CD Deployment

Considering the COVID-19 pandemic, a fast-growing cloud native SaaS client is expanding its product portfolio and as such, they are hiring more developers across the globe that can work remotely in pushing their products to customers as quickly as possible so that they can collect feedback and improve upon the previous iteration of their products. Hence, the need for Jenkins.

Jenkins offers a simple way to set up a continuous integration and continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating several routine development tasks. Even though it does not eliminate the need to script for individual steps, it provides a faster and flexible way to integrate your entire chain of build, test, and deployment tools than you could easily build.

With respect to my client, they wanted a more structured & standard way of provisioning the Jenkins infrastructure such that it is consistent across the entire company.

In addition, my client would like the entire architecture automated so that they can monitor their resources.

Finally, my client wanted to reduce the time DevOps will spend building a CICD pipeline; hence the need to have the deployment via Infrastructure as Code (IaC).

For the project, the Jenkins CI/CD infrastructure & configuration was provisioned using Terraform & Ansible.

Figure #1 is an overview of the Jenkins CI/CD architecture that describes my client's request.

2. What constraints did you have to take into account before deciding what to build?

Some of the constraints that I had to consider before deciding what to build are as follows:

I. Existing layout of their developers or squads. The number of developers will provide an insight on the right-sizing of the EC2 resources. E.g. t3.micro vs t3.large.
II. Operating System platform dictates the number of test servers that I might need to consider in my Jenkins Slave/Worker architecture E.g., Ubuntu vs. Linux vs. Windows
III. DevOp Tools that current developers are familiar with so that they do not spend more time learning new tools that cannot provide additional merits
IV. Type of application that they are developing. E.g., Compute instance vs. General Purpose Instance type.
V. Fault Tolerance & Cost saving measures based on region-to-region traffic E.g. AWS charges $0.02/GB of data transfer from region-to-region except between US-East-1 & US East-2 where it is $0.01/GB.
VI. Resource tagging – Using Cost Explorer, I can generate granular billing reports that are specific to my resources, departments, resource owners etc.. Tagging also provides a great way of identifying and tracking resources.
VII. Using S3 bucket as a backend. The Terraform language is declarative (describing an intended goal rather than the steps to reach that goal), hence; the need to store the state file (tfstate.tf) in a **version enabled S3 bucket** vs. **local hard disk** of the user. E.g., if the user hard disk crashes or there is need to revert to an earlier version of the Terraform State file.
VIII. Security – Need to have AWS Certificate Manager on the port 443 listener of the Application Load Balancer (ALB) so that users request is terminated at the ALB. Also, **http** request on port 80 will be permanently **redirected** to **https** on port 443 before termination.
IX. Security - Ensuring that only allowed ports are enabled via the security groups of the load balancer, Jenkins Master & Jenkins Slave/Worker(s).

3. How did you validate that you were building the right thing?

After meeting with the client or stakeholders to understand what the problem is, I usually follow up with a project charter that describes the project in its entirety.

I then proceed by breaking my deliverables into multiple stages and set a milestone on each stage. With respect to this project, every milestone listed in the "**Wildbit_Jenkins_CICD_Documentation**" is accompanied by provisioning the resource, inspecting and testing via AWS Management Console before adding more resources. If successful, the resource is destroyed to avoid additional charges and the coding continues there after.

Note: For some projects, I will be required to review project with my client at the end of each milestone.

4. What was the process to roll out your solution to your teammates?

The process to roll out my solution to my client/teammates involves the following:

I. A work through process of the "**Wildbit_Jenkins_CICD_Documentation**" with the developers. Here, I might be required to add more features such as adding an Application logic to the documentation or providing a training manual. In some cases, I might be required to provide a transition training to the developers.
II. A review of the requirements for provisioning the infrastructure i.e., operating system, public domain for the certificate manager and route53, installing dependencies like EC2-plugin for Ansible Configuration file, security groups, regions for deployment, VPCs, user authentication via IAM, policies via IAM etc.

5. What was success supposed to look like?

Success on this project is a combination of the following:

1. The project meets the needs of the developers i.e., providing a more consistent Jenkins CI/CD infrastructure for the developers
2. The project is cost sensitive i.e., the project design did consider ways of reducing cloud spend.
3. The project provides an automated way of CI/CD deployment thus allowing the developers focus more on their application versus provisioning the infrastructure.
4. Management can track and monitor CI/CD infrastructure for each product line.