Technical Challenge Assignment

Author	Vasudeva “Vasu” Nayak Kukkundoor
Applied Role	Senior Technical Account Manager, Munich Germany
Assigned Date	11.11.2025
Submitted Date	13.11.2025
Code Repository	https://github.com/vazudew/tech_assessment_jk
Contact	vazudew@gmail.com

Requirements

Please visit the Site for Results, acceptance criteria fulfillment.

(a) The solution must run on a clean installation of the chosen operating system without errors.

Prepared a bash script containing official installation commands from Jenkins to be executed on Ubuntu 24.04. This script is executed error-free on a fresh Instance.

(b) Jenkins and its prerequisites must be installed without manual intervention.

Jenkins prerequisites with Java installation, adding repo/key are all part of the standard script. Implemented them in the installation script.

(c) Jenkins must be configured to serve requests over port 8000.
NOTE: It is not sufficient to forward port 8000 on either the host or the guest OS to the default Jenkins port. Jenkins itself must be configured to listen on port 8000.

Jenkins server is configured to listen on Port 8000. There is no port forwarding or mapping of any sort.

(d) Subsequent applications of the solution should not cause failures or repeat redundant configuration tasks

Solution is idempotent and does not repeat any executions.

Instructions

1. Your code solution in the file format in which it is expected to run.

   The solution is a Terraform/bash script-based automation that deploys an EC2 Instance hosting a Jenkins application on AWS Cloud.

   Please visit this Repository for getting access to the source code, instructions to execute.

2. A brief paragraph with the instructions for running your solution. Please explicitly state all assumptions (e.g., only runs on RHEL)

   Assumptions

   1. The server would be hosted in the *default* VPC, and a Public subnet with a specific tag {function=dev}. This is to ensure EC2 is deployed and can be accessed via the internet. You may have to change the code accordingly for your execution.
   2. The Jenkins server would be running on EC2, with Ubuntu 2404 Server, containing already *AWS SSM Agent* already installed. Other OSes and flavors are not tested and out of scope.
   3. I use a pre-configured SSM Role, which will be set with an *Instance Profile* for the EC2 *Jenkins Server*. The Role would contain *AWS Managed Permissions* such as *AmazonSSMManagedInstanceCore*, *AmazonEC2RoleforSSM*, and *CloudWatchLogsFullAccess*.

      This role enables EC2 Instance (*Jenkins Server*) to be managed directly by *AWS System Manager* for further operations. You may have to change the code accordingly for your execution.

   4. The EC2 Instance has just one *Security Group* attached, to allow inbound traffic from anywhere (0.0.0.0/0) for the only *Jenkins_Port*. Please exercise caution here, as it is not the right way to secure the workload.
   5. As the *Security Group* does not allow *SSH* connections to the EC2 machine, I use *Session Manager* to connect.
   6. The *jenkins_installer.sh* installs the latest *Jenkins* (Version: "2.528.1") binary onto the machine. For additional installation, this script must be updated.
   7. By default, *Jenkins* creates an *admin* user with a password stored in the file */var/lib/jenkins/secrets/initialAdminPassword*. The content of this file has to be fetched to access the *Jenkins* GUI Portal.
   8. For the benefit of time, I have considered a minimalistic approach to focus on the completion of the task.

Execution

Visit this Site for more updated information on execution

Please get a copy of the source code, and you may have to make several changes to execute it on your environment. The source code also has shipped a *Makefile* to run commands to deploy and an *acceptance_test.py* script to validate the deployment.

1. Download the latest project code repo onto your machine

   $ git clone https://github.com/vazudew/tech_assessment_jk

2. Please look at the file _data.tf_ and see various *sources*, those are pre-configured for the solution (also mentioned in Assumptions section). Prepare similar _AWS Resources_ on your Cloud Platform. Modify this file accordingly.

3. Please modify the backend section of config.tf to prepare, store the state files.

4. Prepare the right AWS Credentials, as Terraform needs to communicate with the AWS Account with your credentials

5. Run **make tf-init** to initialize Terraform, downloading providers, and configuring the backend

6. Execute **make tf-plan** to get a holistic view of the deployment plan

7. Once you review the plan and are satisfied, we can move to the deployment of solution. Please run **make tf-apply** command.
8. Please fetch the admin user password by logging into the EC2 instance using AWS Session Manager. The password will be the content of file
   /var/lib/jenkins/secrets/initialAdminPassword

9. You can also use the *acceptance_test.py* script that is shipped with this repo, for ensuring the Jenkins in fact, is accessible, with *admin* user.

   $ python acceptance_test.py --url "http://<Public_IP_Jenkins_Server>:<Jenkins_Port>/" --username admin --password <ADMIN_PASSWORD>

10. Please run **make tf-destroy** to remove entire deployment. Please exercise caution while doing so, as there is no *Backup* or *Disaster Recovery* strategies in place.

Answers

3. Answers to the following questions :

a. Describe the most difficult hurdle you had to overcome in implementing your solution

I really enjoyed the tech challenge. It got me to think and strive always for a better solution.

The difficulty I have realized during task ideation is finding the right balance of declarative and imperative styles of automation. Though I love to have the entire solution deployed in a declarative style, I do not possess the right skills to do so.

However, for the solution, I have used Terraform for resource creation, and user data to execute a bash script to deploy Jenkins. Terraform is a state file-oriented solution and declarative in style and a bash script is imperative and may not work all the time (depends on platforms, versions etc)

My solution deploys Jenkins without any errors or manual efforts. It is also idempotent in nature. However, it lacks to detect any configuration drifts and their remediation.

This is the reason I am very much appreciative of Puppet, Chef and Ansible like tools, to support such use-cases.

Another issue with my solution is that it has a mix of Terraform and bash scripts. Any potential Jenkins management tasks would later involve Groovy scripts. I would love to have one consistent language to work with all aspects.

b. Please explain why requirement (d) above is important.

The statement "*Subsequent applications of the solution should not cause failures or repeat redundant configuration tasks*" is the crux of Infrastructure as Code Concepts.

Golden Rule: The automation must always be reliable, time-saving, reusable, cost-effective, and error-free.

The requirement (d) in this sense means, if the system is already in a desired state (i.e. jenkins is already deployed, accessible on port 8000), our automation solution, upon subsequent execution, must detect that the system does not need any changes and do nothing. However, if there is a change needed in the system, the automation solution must reflect the change, and its execution must bring the system to respective desired state. This concept is idempotency.

Usual scripts (non-idempotent) execute without validating the system and repeat all the execution steps unnecessarily. These executions can be expensive, may introduce a lot of

errors/bugs in the system, causing unreliable behavior, increasing the time to deploy, and thus violating our Golden Rule.

We must always respect the golden rule; this is why requirement (d) is important.

c. Where did you go to find information to help you?

Sl. No.	Link	Remarks
1	Official Jenkins Installation	*bash* script to install pre-requisites, jenkins binaries
2	Jenkins Port Setup	Jenkins configurations and options
3	*override.conf* nuances in Jenkins	Understanding parameter changes in Jenkins
4	*AWS System Manager*	Basics on EC2 config management AWS Style
5	*AWS System Manager Runbook*	executing *Run* Commands on managed EC2 instances
6	*AWS EC2 User Data*	Groundwork for EC2 user data and scripts
7	*Hashicorp Terraform Documentation*	Documentation for various *AWS* resource creation
8	Terraform EC2 Module	easy to use, tested modules for fast deployment
9	Puppet Introduction	Basic understanding of *Puppet* works
10	ChatGPT	Quicker syntax free code samples

d. Briefly explain what automation means to you, and why it is important to an organization's infrastructure design strategy

For me, automation means using technology to execute tasks with no human intervention and thus reducing errors, increasing system reliability, with faster deployment cycles. It can help support the completion of repetitive or predictable tasks proactively and automatically.

An Infrastructure design is core to a company's business goals. It encompasses various components such as Networks, Compute instances, Security Landscapes, Confidential Data, and customers. These factors influence the company, its interests, and customers. Any compromised component, or unavailable/disrupted service, or any major escalation can bring the company's reputation down, and lose all trust from customers.

Therefore, we must ensure our automation for infrastructure design is always reliable, time-saving, reusable, cost-effective, and error-free. Failing this rule, the automation can be counterproductive.

With the right automation strategy, the organization can have the following advantages:

1. **Ensure Consistency**: Infrastructure is deployed, configured the same way every time
2. **Reduces Error**: Minimizes human intervention and hence is less error-prone, and more accurate
3. **Faster Delivery**: The scripts run fast, and hence faster deployment
4. **Improved reliability and recovery**: we can introduce monitoring, self-healing systems to reduce downtimes and validate overall health of the system
5. **Enhanced Scalability, agilit**y: Automation can be reused, parameterized, and hence increase scalability. Also, one can start iteratively and improve automation quality.
6. **Cost-Effective**: Reduce operational costs over time
7. **Standard Industry Practice**: Automation is also part of industry standards ,such as IS= 27001, GDPR and hence your organization is compliant