SOMETIMES THE MOST ELEGANT FIX IS NOT THE MOST ARCHITECTURALLY PURE ONE

Challenges and Solutions

During the implementation of the **Sock Shop Microservices Kubernetes Project**, several technical hurdles were encountered. Below is a documentation of the key issues and the methodologies used to resolve them.

1. Git Conflict: Untracked Local Files

- **The Challenge:** While attempting to synchronize the local repository with the remote main branch (git pull origin main), the process aborted with an error stating that untracked working tree files (specifically module/ansible/playbooks/keepalived.yml) would be overwritten by the merge.
- **The Solution:** To preserve local configuration changes without losing the remote updates, I utilized the **git stash** utility with the --include-untracked flag.
    - **Command:** git stash push --include-untracked -m "preserving local keepalived config"
    - This allowed for a clean git pull, after which I used git stash pop to re-integrate my local changes into the updated codebase.

2. Ansible Syntax: YAML Indentation Errors

- **The Challenge:** A syntax check on the keepalived.yml playbook failed with the error: did not find expected '-' indicator. This occurred because the become: true directive was improperly indented, causing the YAML parser to fail while reading the block collection.
- **The Solution:** Corrected the indentation to ensure that all play-level directives (like hosts and become) were properly aligned as children of the play definition. This reinforced the importance of using ansible-playbook --syntax-check as a standard part of the CI/CD pre-flight routine.

3. Permission Management & Shell Environment

- **The Challenge:** There was initial confusion regarding the use of sudo su -c versus direct sudo commands for administrative tasks.
- **The Solution:** Research was conducted to implement Linux best practices. I transitioned to using sudo -i for full login shell environments and sudo -u <user> for running commands as specific service accounts (like www-data). This

ensures that the correct environment variables (like $PATH and $HOME) are loaded, reducing "command not found" errors during automation.

4. Kubernetes Observability & Debugging

- **The Challenge:** Distinguishing between infrastructure-level failures (Nodes) and application-level crashes (Pods) during the initial cluster bootstrap.
- **The Solution:** Developed a systematic troubleshooting workflow using kubectl:
    - **Infrastructure Check:** Verified server health using kubectl get nodes.
    - **Application Check:** Monitored the microservices status using kubectl get pods.
    - **Log Inspection:** Utilized kubectl logs -f <pod_name> to stream real-time standard output, allowing for the immediate identification of application-level runtime errors.

5. Network Partitioning: Weave Net CNI Port Blockage

- **The Challenge:** After successfully initializing the Kubernetes cluster and deploying the **Weave Net** Container Network Interface (CNI), I observed that while the Nodes were marked as Ready, Pods located on different Worker Nodes were unable to communicate with each other. This caused microservices like the "Socks Catalog" and "Front-end" to fail their health checks.
- **The Root Cause Analysis:** By inspecting the Weave Net Pod logs using kubectl logs, I identified that the "gossip protocol" used by Weave to build the network topology was being timed out. I realized that the **AWS Security Groups** (the virtual firewalls) were blocking the specific ports required for the CNI overlay network.
- **The Solution:** I modified the AWS Security Group associated with the Kubernetes Cluster nodes to allow **Inbound** and **Outbound** traffic on the following ports:
    - **TCP 6783:** For the Weave control plane.
    - **UDP 6783 & 6784:** For the Weave data plane (actual container traffic).
- **The Result:** Once the ports were opened, the Weave Net nodes successfully established a "mesh" connection. Pod-to-Pod communication was restored immediately, and the microservices application became fully functional across the entire multi-node cluster.

6. Jenkins Pipeline: Binary Architecture Mismatch (OS Error)

- **The Challenge:** During the initial execution of the Jenkins pipeline, the build failed immediately at the infrastructure provisioning stage. The terraform init command triggered a "Segmentation Fault" or failed to execute entirely, halting the CI/CD process.

- **The Root Cause Analysis:** Upon investigating the Jenkins Global Tool Configuration, I identified a platform mismatch. I had accidentally configured the Terraform plugin to download the FreeBSD (amd64) binary instead of the Linux (amd64) binary required by the Ubuntu-based Jenkins agent.
- **The Solution:** I corrected the Jenkins configuration by navigating to **Manage Jenkins > Global Tool Configuration** and updating the Terraform installer settings to the correct OS-specific version: Terraform 1.14.3 linux (amd64).
- **The Result:** After the correct binary was applied, Jenkins was able to properly initialize the Terraform backend. The pipeline proceeded to successfully provision the AWS VPC and subnets, establishing the foundation for the rest of the deployment.

7. sh destroy.sh not deleting the s3 bucket

The challenge: The error says, jq command not found. Thus, looking at the sh destroy,sh syntax on line 25, there is a jq command to be use but the jq plugin is clearly not available on my cli.

Solution:

Download jq to home directory first using command below-

curl -L -o ~/jq.exe https://github.com/jqlang/jq/releases/latest/download/jq-win64.exe

Make it executable and test-

chmod +x ~/jq.exe

~/jq.exe --version

Add to PATH permanently (user directory)

echo 'export PATH="$HOME:$PATH"' >> ~/.bashrc

source ~/.bashrc

verify that jq now works globally

jq --version

8. Setting up Profile issues due to other projects run on the VS code:

Challenge: the AWS profile does not match the profile I intend to use. I need VS Code per-project AWS profile handling. I need each VS Code workspace use its own AWS profile automatically (e.g., pet_team, sock_shop, etc.), especially for integrated terminals.

Solution:

A. Open your project in VS Code

B. Ctrl+Shift+P → "Preferences: Open Workspace Settings (JSON)"

C. If .vscode folder doesn't exist → VS Code creates it automatically

D. Paste this content:

```
{
  "terminal.integrated.env.windows": {
    "AWS_PROFILE": "pet_team"
  },
  "terminal.integrated.defaultProfile.windows": "Git Bash"
}
```

REPEAT FOR sock_shop

9. Pushing the codes to my personal repo:

Challenge: To push the project codes to my personal repo.

Solution:

- create your repo (just create a basic private or public repo of your choice, do not add a readme at this level)

- then, run this on the CLIof the project you are working on

git remote add personal https://github.com/YOUR_USERNAME/your-new-repo.git

verify it worked via running this command: git remote -v

After verification, then run to push the current feature branch to your github repo as the main branch: git push personal feature/sock-shop-k8s:main

10. Terraform + Route 53 challenges

Challenge: no matching Route 53 Hosted Zone found. data "aws_route53_zone" "my-hosted-zone" was using a placeholder or incorrect name that did not match any hosted zone in the selected AWS account/profile.stackoverflow+1

Solution: on the CLI

1. List hosted zones for the relevant profile:

aws route53 list-hosted-zones --profile sock_shop

2. Use either the correct name is used directly or as a variable

```
#domain
variable "domain_name" {
  default = "aleyi.space"
}
```

7. Since the project uses variable, that's the preferred solution and when effected, it worked seamlessly.

11. No ACm Certificate

Challenge: No ACM certificate was found

Solution:
Create Certificates Manually on the AWS console (Recommended)

1. AWS Console → ACM (us-east-1) → Request certificate

Type in your Domain: *.yourdomain.com (aleyi.space)

Use DNS Validation: DNS → Creates CNAME records for your Route53(ensure you had already created a hosted zone for the domain name)

Wait for ISSUED status (5-30 mins)

12. Terraform lock state configuration

Challenge: Terraform is STILL configured for sock-shop-team33 - it hasn't been updated to your bucket sock-shop-teamz33.

Solution:
grep -r "sock-shop-team33"

grep -r "team33"

grep = "Find where the problem is hiding" → Edit that file → Fixed forever!

Then run:

rm -rf .terraform/

The use: Completely deletes Terraform's temporary files (like downloaded providers, cached config). Think of it as "Factory reset for Terraform".

Use the command below to create the new named bucket

aws s3api create-bucket \

--bucket sock-shop-teamz33 \

--region us-east-1

Then run:

terraform init -reconfigure

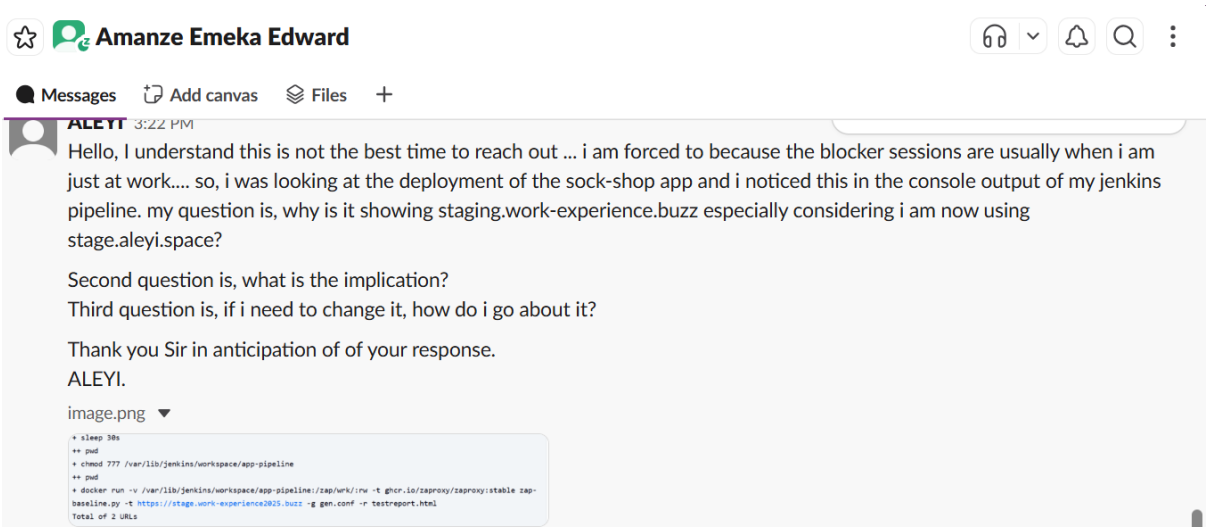
The use: These commands ensure Terraform forgets the old team33 config completely and uses your new bucket name.

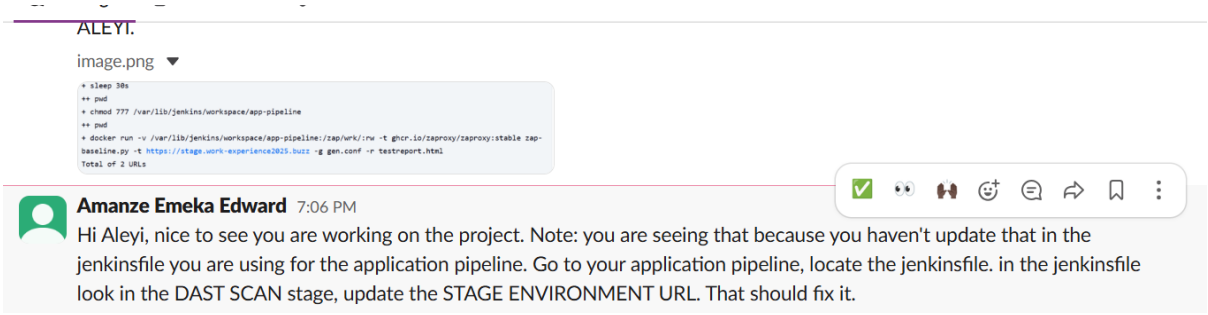
13. **Jenkins Pipeline GitHub Push Permission Denied (403 Error)**

Challenge: Jenkins uses ALEYI001 credentials to push to CloudHight/Sock-Shop-App-Repo.git, but ALEYI001 lacks write access to CloudHight's repository.

Solution: i went into the main and stage jekinsfiles of the application repo and updated the url there directly to ALEYI001/Sock-Shop-App-Repo.git

14. The Attached image details this challenge and the solution. I escalated to my team lead, and he sign-posted me correctly to solution.





15. Inability to delete sock-shop-bastion-role and sock-shop-ansible-role

Challenge: Failed deleting role sock-shop-bastion-role and sock-shop-ansible-role. Cannot delete entity, must remove roles from instance profile first. sock-shop-bastion-role and sock-shop-ansible-role, still linked to IAM instance profile

Solution: on the CLI,

Confirm instance profile:
aws iam list-instance-profiles-for-role --role-name sock-shop-bastion-role

Remove the role from the profile:
aws iam remove-role-from-instance-profile \

--instance-profile-name sock-shop-bastion-role \

--role-name sock-shop-bastion-role

Retry terraform destroy:

terraform destroy -target=aws_iam_role.bastion_role -lock=false

terraform destroy -lock=false

The Result: Full cleanup and the roles will no longer be a problem.

16. The attached image shows the error, and I escalated via the blocker session

```
Error: importing EC2 Key Pair (utility2-keypair): operation error EC2: ImportKeyPair, https response error StatusCode: 400,
RequestID: 4400d74f-14c7-46d2-9958-707f19a258e6, api error InvalidKeyPair.Duplicate: The keypair already exists

   with aws_key_pair.public_key,
   on main.tf line 22, in resource "aws_key_pair" "public_key":
   22: resource "aws_key_pair" "public_key" {


Error: creating IAM Instance Profile (utility2-Jenkins-profile): operation error IAM: CreateInstanceProfile, https response
error StatusCode: 409, RequestID: 085ee27e-befd-48dd-9fb3-eb67af292ceb, EntityAlreadyExists: Instance Profile utility2-Jenkins
-profile already exists.

   with aws_iam_instance_profile.jenkins_instance_profile,
   on main.tf line 104, in resource "aws_iam_instance_profile" "jenkins_instance_profile":
  104: resource "aws_iam_instance_profile" "jenkins_instance_profile" {
```

On the blockers session lead by Mr. Emeka, he gave me too options considering that the resources in question are not costing me.

a. Rename the resources on my code so that a fresh one can be created.
b. Go to my AWS console and delete the resources from there.

I went with the first option and that fixed the issue.