2. Protect your platform resources with a service control policy

Use case:

A central platform team often deploys a standard set of AWS resources when provisioning a new account. These resources are often security related or control some other operational aspect of the platform and should not be modified or deleted, except by privileged IAM principals.

Policy type - Service control policy

Values to use in policy:
Tag on resources to protect:
Key: team
Value: admin
Tag on principals that are exempt:
Key: team
Value: admin
Resource: all resources
Action: all actions

The Approach

1. **Identifying the Right Condition Keys:**

For the implementation of this policy, I utilized two AWS **condition keys**: aws:ResourceTag/team and aws:PrincipalTag/team. These keys are instrumental in verifying both the tags on the resources being accessed and the tags associated with the IAM entities making the requests. This dual verification ensures that resources tagged with team: admin are only accessible to principals similarly tagged, aligning with our organizational access controls.

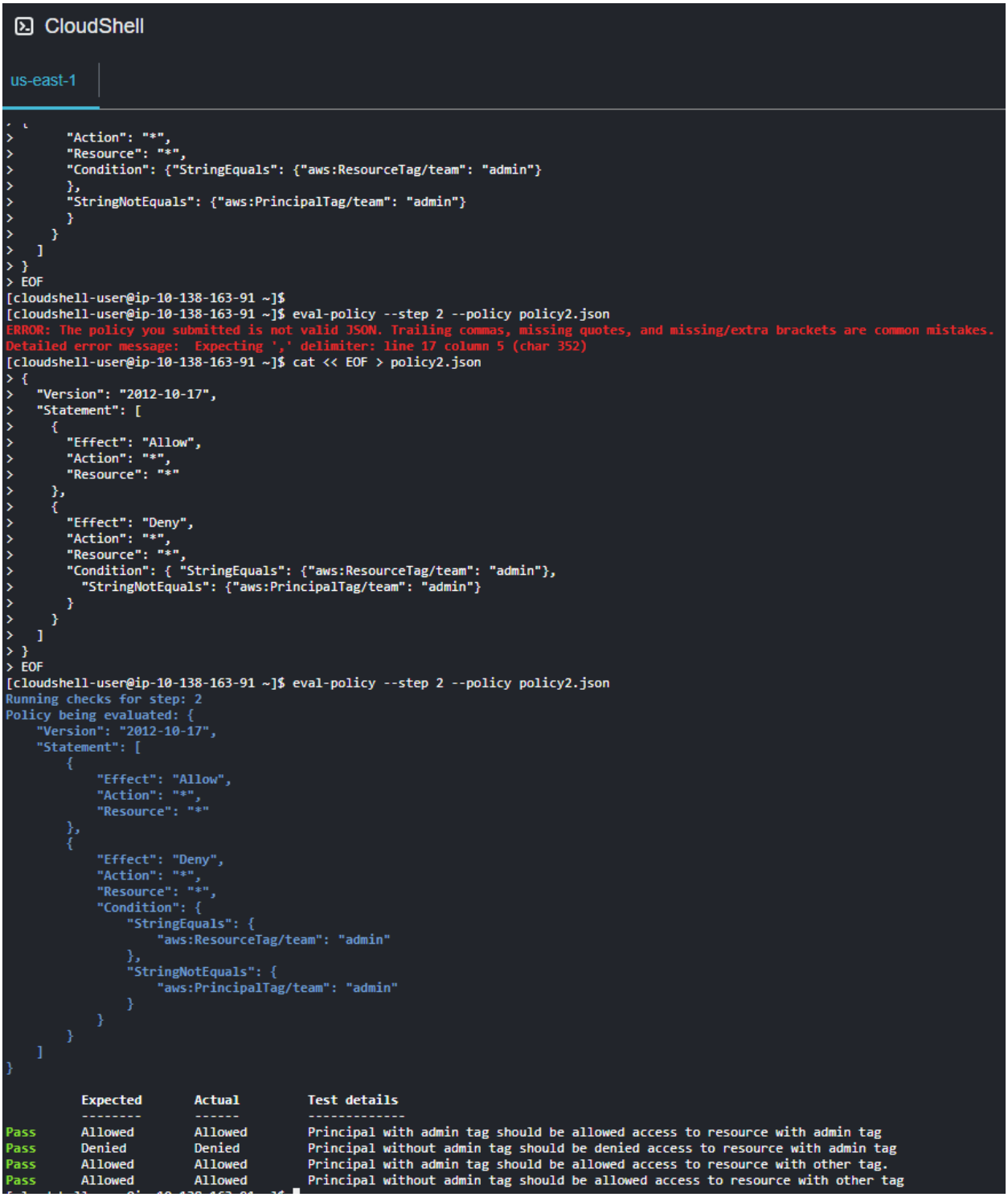
2. **Understanding Policy Elements:**

Principal: Set to * to apply the policy to all IAM roles/users within our AWS account.
Action: Specified as *, which encompasses all possible actions within the AWS environment, allowing for a broad control scope.
Resource: Also set to *, ensuring that the policy is applicable to all resources. This global setting is crucial for a service control policy that needs to enforce comprehensive access restrictions across all AWS services and resources.
Condition:
StringEquals: Ensures that the action is only allowed if the resource being accessed is tagged with key of team and a value of admin.

StringNotEquals: Ensures that the action is denied if the principal attempting the action is not tagged with a tag that has a key of team and value of admin. This condition is pivotal in safeguarding sensitive resources by restricting access to authorized personnel only.

```
cat << EOF > policy2.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": { "StringEquals": {"aws:ResourceTag/team": "admin"},
        "StringNotEquals": {"aws:PrincipalTag/team": "admin"}
      }
    }
  ]
}
EOF
```

The execution:



Testing and validation

Evaluation Command: Ran `eval-policy --step 2 --policy policy2.json` to rigorously test the policy's effectiveness in a controlled environment.

The Outcome

1. **Policy Deployment:** Once validated, the policy was deployed across the necessary AWS accounts within our organization.
2. **Enhanced Security:** The implemented SCP effectively restricted access to critical resources, ensuring that only IAM principals with the team: admin tag could modify the resources tagged similarly.

Challenges:

Addressing JSON Format Issues:

The first draft included errors in the JSON formatting, particularly in the conditional blocks. There were missing commas and misplaced brackets which were causing syntax errors. This was a crucial step as incorrect JSON syntax could prevent the policy from being processed correctly.

Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {"StringEquals": {"aws:ResourceTag/team": "admin"}
      }

      {"StringNotEquals": {"aws:PrincipalTag/team": "admin"}
      }
    }
  ]
}
```

After some revisions and validation against JSON syntax validators, the correct format was established with properly placed conditional operators and attributes.

Policy Refinement:
Adjusted the policy to explicitly include conditions that matched the organizational security requirements:

```
  "Condition": { "StringEquals": {"aws:ResourceTag/team": "admin"},
     "StringNotEquals": {"aws:PrincipalTag/team": "admin"}}
```

Reflections:

This project highlighted the importance of meticulous attention to detail in policy creation, especially the precision required in JSON formatting and the necessity for thorough testing. It reinforced the critical role that well-crafted SCPs play in maintaining a secure and compliant AWS environment.