**Password Less Authentication:**

**Commands**: These all commands need to be performed on Control Node or the one who wants to connect with other ec2.

1. **ssh-keygen -t rsa -m PEM - generate private key file**
2. **ls -lah ~/.ssh – check the permission of the file**
3. **cat ~/.ssh/id\_rsa.pub – copy the generated file content**
4. **cp /mnt/o/awsKeys/ansible-ubuntu.pem ~/.ssh/ - copy the ec2 keypair to ssh folder**
5. **ls -lah ~/.ssh – check permissions**
6. **chmod 400 ~/.ssh/ansible-ubuntu.pem – change permissions**
7. **ls -lah ~/.ssh – check permissions**
8. **cat ~/.ssh/id\_rsa.pub | ssh -i ~/.ssh/ansible-ubuntu.pem ubuntu@3.110.156.85 "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized\_keys" – copy the private and public keys to ec2**
9. **ssh -i ~/.ssh/ansible-ubuntu.pem ubuntu@3.110.156.85 "cat ~/.ssh/authorized\_keys" – check if they are copied to ec2**
10. **ssh** [**ubuntu@3.110.156.85**](mailto:ubuntu@3.110.156.85) **– connect to ec2**

**Notes**:

1. If you are on windows, you might encounter the issue - No identities found. This is because, ssh keys are not available on your machine. You can create them using ssh-keygen -t rsa
2. If you get any permissions error, try following commands.
   1. Copy to another loaction: cp /mnt/o/awsKeys/ansible-ubuntu.pem ~/.ssh/
   2. Change Permissions: chmod 400 ~/.ssh/ansible-ubuntu.pem
   3. Check Permissions: ls -l ~/.ssh/ansible-ubuntu.pem
   4. Output: -r-------- 1 krupa krupa 1678 Jul 8 05:39 ~/.ssh/ansible-ubuntu.pem
3. Link for any Reference: <https://linuxbeast.com/blog/how-to-setup-passwordless-ssh-login-on-ec2-ubuntu-22-04/>

**Ansible**

**Installation:** Toinstallansible in ubuntu(my case) we need python3 and pip package. So, all the steps are as follows…

1. Apt Update
2. Apt Upgrade
3. PIP get: curl [https://bootstrap.pypa.io/get-pip.py -o get-pip.py](https://bootstrap.pypa.io/get-pip.py%20-o%20get-pip.py)
4. PIP Install: python3 get-pip.py –user
5. Install Virtual Environment Package: apt install python3-venv
6. Create Python Evnironment: python3 -m venv ansible-env
7. Activate Environment: source ansible-env/bin/activate
8. Install ansible: pip install ansible

**Note**: Once you exited the IDE/VM you hade to go to the ansible environment again to work with ansible. Follow the steps below:

1. Go to path where you store the env: cd ~/ansible-env
2. Activate ansible: source bin/activate

**AdHoc** **Commands**:

1. Ansible -I inventory.ini -m ping all
2. Ansible -I inventory.ini -m shell -a “sudo ls /etc/” all
3. Ansible-playbook -I inventory.ini “playbook.yaml”
4. ansible-vault create group\_vars/all/pass.yml --vault-password-file vault.pass
5. Ansible-playbook -I inventory.ini “playbook.yaml” –vault-password-file “password file”
6. Ansible-playbook playbook,yaml –vault-password-file ‘password file’

**Roles**:

1. Roles are like designations which are given to different users for different purposes.
2. With the roles we can use variable precedences effectively, however we like.
3. By using roles from ansible-galaxy we can do lots of easily due to that in that role, there is an yaml file which we can use to do all those required tasks.
4. Example: We can install all docker and its components by getting a role from ansible galaxy, if not we have to write an yaml file which takes hours of our valuable time.
5. For more, visit the link: <https://galaxy.ansible.com/ui/>

**Yaml** **File**:

1. Use “cat > ‘filename’ “ to create and paste the yaml content to local machine.
2. When editing the existing yaml file, use ‘vim’ command.
3. Use variables to ease the workflow, don’t do hard coding all the time.
4. Check the varialble precedence documentation whenever you are in need by visiting this link:[https://docs.ansible.com/ansible/latest/playbook\_guide/playbooks\_variables.html#variable-precedence-where-should-i-put-a-variable](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html%23variable-precedence-where-should-i-put-a-variable)

**Vault**:

1. Store sensitive info like AWS Keys in file which will be protected by vault using a generated password. The commands are as follows…
   1. Create Password: openssl rand -base64 2048 > vault.pass
   2. Create File: ansible-vault create group\_vars/all/pass.yml --vault-password-file vault.pass
   3. Edit File: ansible-vault edit group\_vars/all/pass.yml --vault-password-file vault.pass
   4. Custom Password: ansible-vault create ‘FileName’ => give your own password
   5. To see Passwords with ‘cat’ command, you need to first ‘decrypt’ it with this:

Ansible-vault decrypt ‘Filemname’ –vault-password-file ‘vault.pass’

* 1. Don’t forget to encrypt it afterwards.

**Amazon** **AWS** **using** **Ansible** - **Galaxy**

To use AWS functionalities like ec2, s3 and more, we need to install some packages like boto3, amazon packages first.

1. Install Boto3: pip install boto3 / apt install python3-boto3
2. Install AWS Collections: ansible-galaxy collection install amazon.aws
3. To know more about commands for AWS, go to this link: <https://docs.ansible.com/ansible/latest/collections/amazon/aws/index.html>

**Notes**:

1. Ansible is Idempotent, so, It won’t do the same task again, intead it will skip that task.
2. Ansible Github Help: <https://github.com/iam-veeramalla/ansible-zero-to-hero>

**Terraform**

1. Terraform runs with Hashi Corp Language (HCL) and its owned by Terraform.
2. Terraform can work with any other cloud platforms such as Azure, GCP, Openstack, AWS.
3. Universal Approach – API as a Code (AAAC)
4. We need AWS CLI to be installed on our system for terraform to work.
5. AWS CLI Installation: <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>
6. Terraform Installation Video: <https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli>
7. Terraform files extension is ‘tf’. Ex: main.tf
8. Terraform AWS Manual: <https://registry.terraform.io/providers/hashicorp/aws/latest/docs>
9. Terraform Github Help: <https://github.com/iam-veeramalla/terraform-zero-to-hero>
10. Install the required extension to VsCode: Terrafor, HashiCopr Terraform, HCL

**Terraform Commands:**

1. Initialize: terraform init
2. Test: terraform plan
3. Execute: terraform apply
4. Terminate: terraform destroy
5. View State File: terraform show
6. Create Workspace: terraform workspace new ‘workspace\_name’
7. Switch Workspaces: terraform workspace select ‘workspace\_name
8. Give the custom tfvar file: terraform apply –var-file=’file name’

**Vault**:

1. Install gpg : sudo apt update && sudo apt install gpg
2. Download the signing key to a new keyring : wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
3. Verify the key's fingerprint : gpg --no-default-keyring --keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg –fingerprint
4. Add the HashiCorp repo : echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
5. Update Packages : sudo apt update
6. Install Vault : sudo apt install vault
7. Start Vault : vault server -dev -dev-listen-address="0.0.0.0:8200"
8. Open the url site Vault : “public\_ip\_address:8200” and login using the token.
9. Enable Approle or any Role
10. To enable the AppRole authentication method in Vault, you need to use the Vault CLI or the Vault HTTP API.
11. Open another Instance and run this cmd and you can get this when you do Step7: export VAULT\_ADDR='http://0.0.0.0:8200'
12. Create Policy :

vault policy write terraform - <<EOF

path "\*" { capabilities = ["list", "read"] }

path "secrets/data/\*" { capabilities = ["create", "read", "update", "delete", "list"] }

path "kv/data/\*" { capabilities = ["create", "read", "update", "delete", "list"] }

path "secret/data/\*" { capabilities = ["create", "read", "update", "delete", "list"] }

path "auth/token/create" { capabilities = ["create", "read", "update", "list"] }

EOF

1. Create the AppRole:

vault write auth/approle/role/terraform \

secret\_id\_ttl=10m \

token\_num\_uses=10 \

token\_ttl=20m \

token\_max\_ttl=30m \

secret\_id\_num\_uses=40 \

token\_policies=terraform

1. Generate Role ID: vault read auth/approle/role/’policy-role-name’/role-id
2. Generate Secret ID: vault write -f auth/approle/role/’policy-role-name’/secret-id

**AWS**

* Github Repo Link: <https://github.com/iam-veeramalla/aws-devops-zero-to-hero>
* YT Playlist : <https://www.youtube.com/playlist?list=PLdpzxOOAlwvLNOxX0RfndiYSt1Le9azze>
* Tutor: Abhishek Veeramalla
* AWS is a Public Cloud and it is Pay as you go Model.
* AWS is best among other cloud platforms such as Azure, GCP, Oracle, Alibaba …
* It started with Virtual Machine methods.
* Kubernetes is the most popular one now.
* AWS is the first one in the cloud market like older than others.
* More job opportunities in AWS than other and the order is AWS > Azure > GCP.
* **Services:** EC2, RDS, CloudWatch, IAM, VPC, Route53, S3, AutoScaling, Load Balancer, Terraform, Ansible, Git, GitLab, GitHub, Lambda, Kubernetes, Linux, Ubuntu, EKS, EFS, Cost Optimisation, CLI, Grafana, Prometheus.

**IAM**

* **IAM** – Identity Access Management (Manage Access to AWS Resources)
* **Terms:** Users, Groups, Policies, Roles, Terms & Conditions, Access Key, Secret Key, CLI, Permissions, Restrictions, Authenication, Authorization, Username, Password.
* **AWS 🡺 Security 🡺 IAM 🡺 Access Management 🡺 Follow Onscreen Intructions.**
* **Policy Types:** AWS Managed Policies, Custom Policies,
* **Policies:** IAMUserChangePassword, AmazonS3FullAccess, AmazonDMSRedshiftS3Role,
* Policies are in JSON Format.
* Groups are nothing but Teams in which all the users need same policy for their roles.
* Root User is Admin or Primary User of AWS.

**EC2**

* **Ec2** – Elastic Cloud Compute (Virtual Servers in the Cloud)
* **Elastic –** Scale UP and DOWN
* **Terms:** Instances, Regions, Instance Types, AMI, Linux, Ubuntu, Windows, Key Pairs, VPC, Subnet, Public IP, Security Groups, Ports, Availability Zones, Monitoring, CPU, Storage, RAM, Machine, Image, Termination, Stop, Start, Shell Scripting, Load Balancers, Auto Scaling, EBS and more…
* **AWS 🡺 Compute 🡺 EC2**

**Load Balancer**

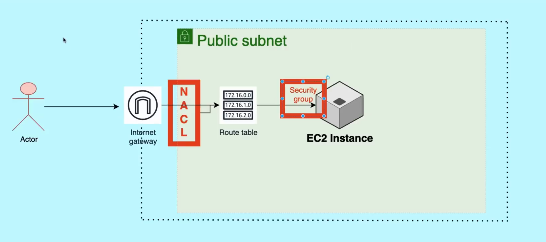
* **Load Balancer** – It distributes capacity to different instances in the Target Groupd during peak traffic times, and increases the reliability of applications.
* **Terms:** Auto Scaling Group, Instance, Launch, Terminate, Traffic, Stress, Target Groups
* **AWS 🡺 Compute 🡺 EC2 🡺 Load Balancers**
* **Target Group 🡺 Target Type (Instance) 🡺 Name 🡺 Protocol 🡺 VPC 🡺 Health Check 🡺 Select Instances 🡺 Include as Pending 🡺 Create Target Group**
* **Load Balancers 🡺 Create Load Balancer 🡺 Load Balancer Type 🡺 Name 🡺 Scheme 🡺 Network 🡺 VPC 🡺 Sunets - Availability Zones 🡺 Security Group 🡺 Select Target Group 🡺 Create Load Balancer**
* Target Gropu is a Collections of Instances that a Load Balancer Can work on

**Auto Scaling**

* **Auto Scaling** – Scalable Instances by Launcing new ones and Terminating in need.
* **Terms:** Auto Scaling Group, Instance, Launch, Terminate, Load Balancer, Stress
* **AWS 🡺 Compute 🡺 EC2 🡺 Auto Scaling**
* **AutoScaling Groups 🡺 Name 🡺 Launch Template 🡺 Create Template 🡺 EC2 Instance Details 🡺 Select Launch Template 🡺 Next 🡺 Select VPC 🡺 Availability Zones & Subnets 🡺 Load Balancing 🡺 Group Size 🡺 Scaling Policies 🡺 Notifications 🡺 Launch**
* Auto Scaling Group is a collection of EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management.
* Don’t select VPC in Launch Template.

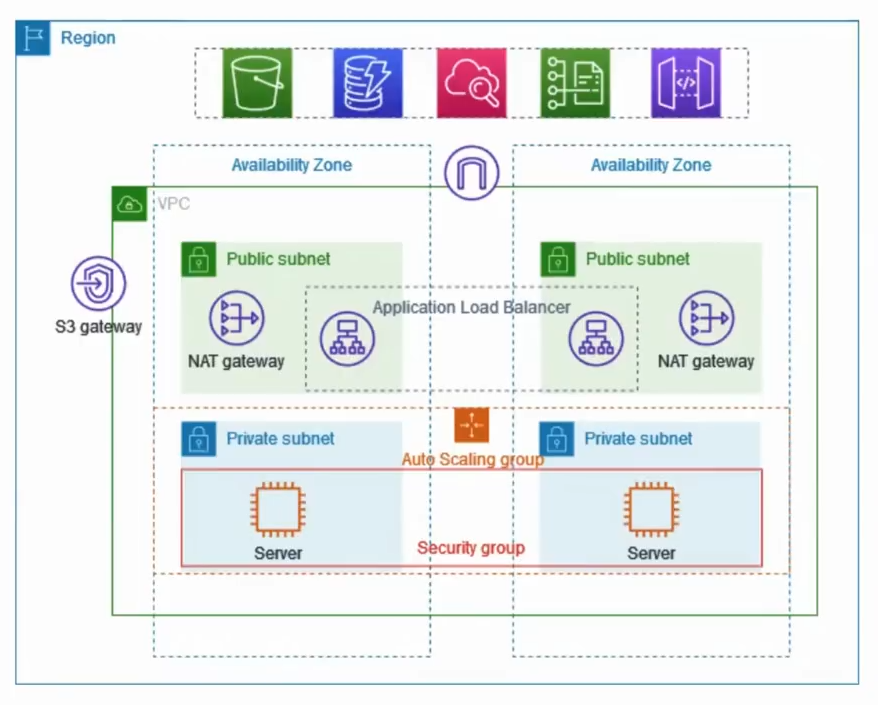
**VPC**

* **VPC** – Virtual Private Cloud(Isolate Cloud Resources)
* **Terms:** Public Cloud, Private Cloud, IP Address, CIDR, Subnets, Route Table, Public Subnet, Private Subnet, Internet Gateway (IGW), NAT Gateway, Security Group, Elastic Load Balancer, NACL, Masking of IP, SNAT, DNS, S3 Gateway, Route 53, Elastic IPs
* **AWS 🡺 Network & Content Delivery 🡺 VPC**
* Without VPC, AWS security is not that good.
* Security is the main thing when it comes to VPC.
* **SG** – Security Group Serves at the Instance Level.
* **SG terms** – Inbound trafic (Ingress), outbound traffic (Egress)
* **NACL** – Network Access Control List
* **NAT** – Network Address Translation
* NAT Gateway masks the ip address when doing api calls.
* SG & NACL acts as a last point in the scurity.
* Elastic Ips are Static IP’s, won’t change.
* **Flow (Figure 1):** **User 🡺 IGW 🡺 Public Subnet 🡺 NACL 🡺 Route Table 🡺 SG 🡺 EC2 Instance**
* Route Table controlls the traffic flow in the VPC.



**Figure 1 : VPC - Public Subnet**

* **Flow (Figure 2):** **User 🡺 IGW 🡺 VPC 🡪 AZ 🡪 Public Subnet, Private Subnet 🡪 NAT Gateway 🡺 Auto Scaling 🡪 Auto Scale Group 🡪 Instance (private) 🡺 EC2 🡺 Bastion Host (public) 🡪 Connect other Private Instances 🡪 Deploy Application in the Private Instances Using Bastion Host 🡺 Load Balancers 🡪 Add Private Instances 🡺 Complete**
* **Add Required Ports in Security Groups Inbound and Outbound Rules.**



**Figure 2: VPC FLOW Public Private Subnets**

**Bastion Host or Jump Server**

* **Bastion Host** – Provides access to a private network from an external network, such as the Internet.
* **Terms:** VPC, Traffic, SSH, Internet Access
* **AWS 🡺 Network & Content Delivery 🡺 VPC**
* It is a bridge between public and private networks.
* Bastion Host or Jump Server is nothing but another Instance which can used as a mediator between Public and Priveate Networks.
* **Connect to Bastion:**
  + **Copy Key:** scp -I “key path” “key path” “user@ipaddress”:”path”
    - Ex: scp -I ansible-ubuntu.pem ansible-ubuntu.pem user@public\_ip: /home/ubuntu
  + **Connect:** ssh -I ansible-ubuntu ubuntu@public\_ip
* **Connect to Private Instance from Bastion Host:** ssh -I ansible-ubuntu ubuntu@public\_ip

**Route 53**

* **Route53** – Scalable DNS and Domain Name Registration
* **Terms:** DNS, Web Hosting, Domain, Server, Aname, Cname, IP Address Mapping, Load Balancer, Hosted Zones, DNS Records, Web Server, Health Checks
* **AWS 🡺 Network & Content Delivery 🡺 Route 53**
* **DNS – Domain Name System**
* IP Addresses can be Static and Dynamic.
* DNS keeps a number of records.

**S3**

* **S3** – Simple Storage Service (Scalable Storage in the Cloud)
* **Terms:** Bucket, 11’9’s, Scalable, Highly Available, Cost Effective, Security, Performance, Data, Any File Type, Databases, Backup, Storage, HTTP Protocall, Global Acces, Bucket Policies, Snapshots, Object Lock, Static Website Hosting
* **AWS 🡺 Storage 🡺 S3 🡺 Create Bucket 🡺 Global Unique Name 🡺 Region 🡺 Ownership (ACL) 🡺 Block Public Access 🡺 Bucket Versioning 🡺 Encyption 🡺 Create Bucket**
* **S3 Reliability Score – 99.99999999999 (11’9’s)**
* **5 Major Advantages:** Availability & Durability, Scalability, Security, Cost Effective, Performance.
* **ACL :** Access Control List
* One Object should not be more than 5 TB.
* Cost will be depends on the Storage Class we Choose.
* Bucket Versioning will store versions of the file and can be work as a Version Control System.

**CLI**

* **CLI** – Command Line Interface
* **It acts as a layer between User and AWS**
* **Terms:** Commands, Shell, Scripting, API, Automation, IAC, Access Key, Secret Key, Configure, Cloud Formation Templates, JSON, Parameters, Create Instances, S3…
* **CLI 🡺 Python 🡺 AWS**
* **ACL :** Access Control List
* **API:** Application Programming Interface
* **Alternatives:** Terraform, Ansible, CFT
* UI is not automation friendly so the CLI is used wherever it required.
* CLI is best for Quick Use Case Thing or Simple Actions like Getting The Data.
* CLI is not best for Infrastructure creation (IAC)
* **Documentation:** <https://docs.aws.amazon.com/cli/latest/reference/>

**CFT**

* **CFT** – Cloud Formation Template (Create and Manage Resources with Templates)
* **Terms:** Creating, Managing, IAC, Infrastructure, CLI, API Calls, YAML, JSON, Drift Detection, Stacks, Parameters
* **Work Flow: User 🡺 CFT 🡺 AWS**
* **AWS 🡺 Management & Governance 🡺 Cloud Formation 🡺 Create Stack 🡺 Template 🡺 Upload 🡺 Create**
* **IAC:** Infrastructure As Code
* CFT Only supports AWS
* CFT Supports YAML and JSON Languages.
* CFT is best for Putting the Data than CLI like Creating Resources and more.
* CFT can be created by CLI or UI.
* Drift Detection shows the changes made to the created resources through CFT.
* **Documentation:**https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-guide.html
* **Tempate References:** <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-reference.html>
* **Prerequisites:** VS Code Editor, YAML Plugin, AWS Toolkit Plugin
* If auto suggestions on vscode not showing, try changing the scema at the bottom right corner to AWS cloud formation..

**IAC**

* **IAC** – Infrastructure As Code
* IAC Providers need to act as a mediator between Any Cloud Provider and User.
* **Terms:** Creating, Managing, Infrastructure, Mediator, Automation, API Calls, Declarative, Versioned, Git, Templates, YAML, JSON
* **Example:** Terraform, Ansible
* **JSON:** Can’t put Commetns, Works with Bracket Mechanism.
* **YAML:** Can put Comments, Works with Indentation Mechanism.
* **IAC:** User writes a Code to create the Infrastructure
* **Declarative:** What you see is What you have and all the code is local.
* **Versioned:** Gets Part / Full Codes from Git Repository or any online cloud platform.

**CICD**

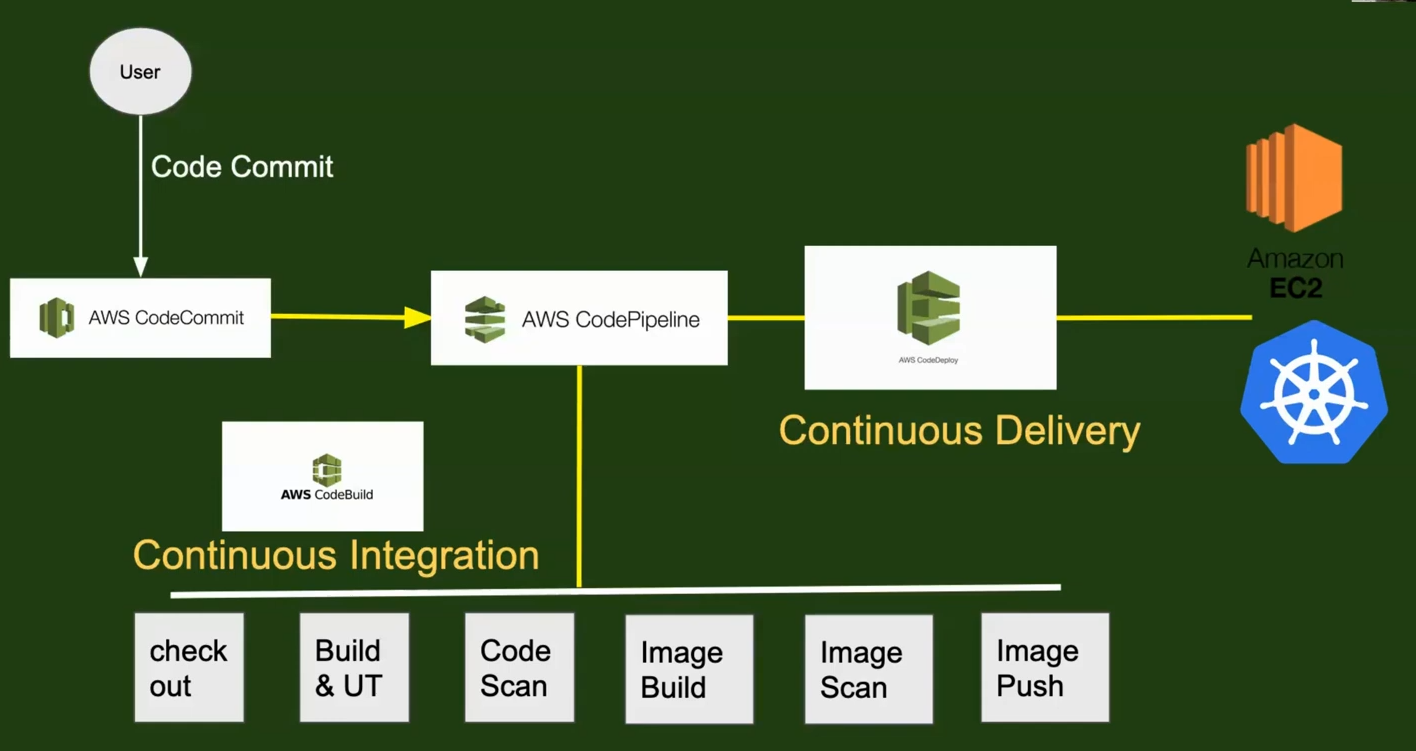
* **CICD** – Continuous Integration Continuous Deployment
* **Services:** AWS CodeCommit (GitHub), AWS CodePipeline (Jenkins), AWS CodeBuild (Maven), AWS CodeDeploy (Ansible/Terraform), AWS CodeArtifact (Maven Artifacts)

**1.AWS Code Commit**

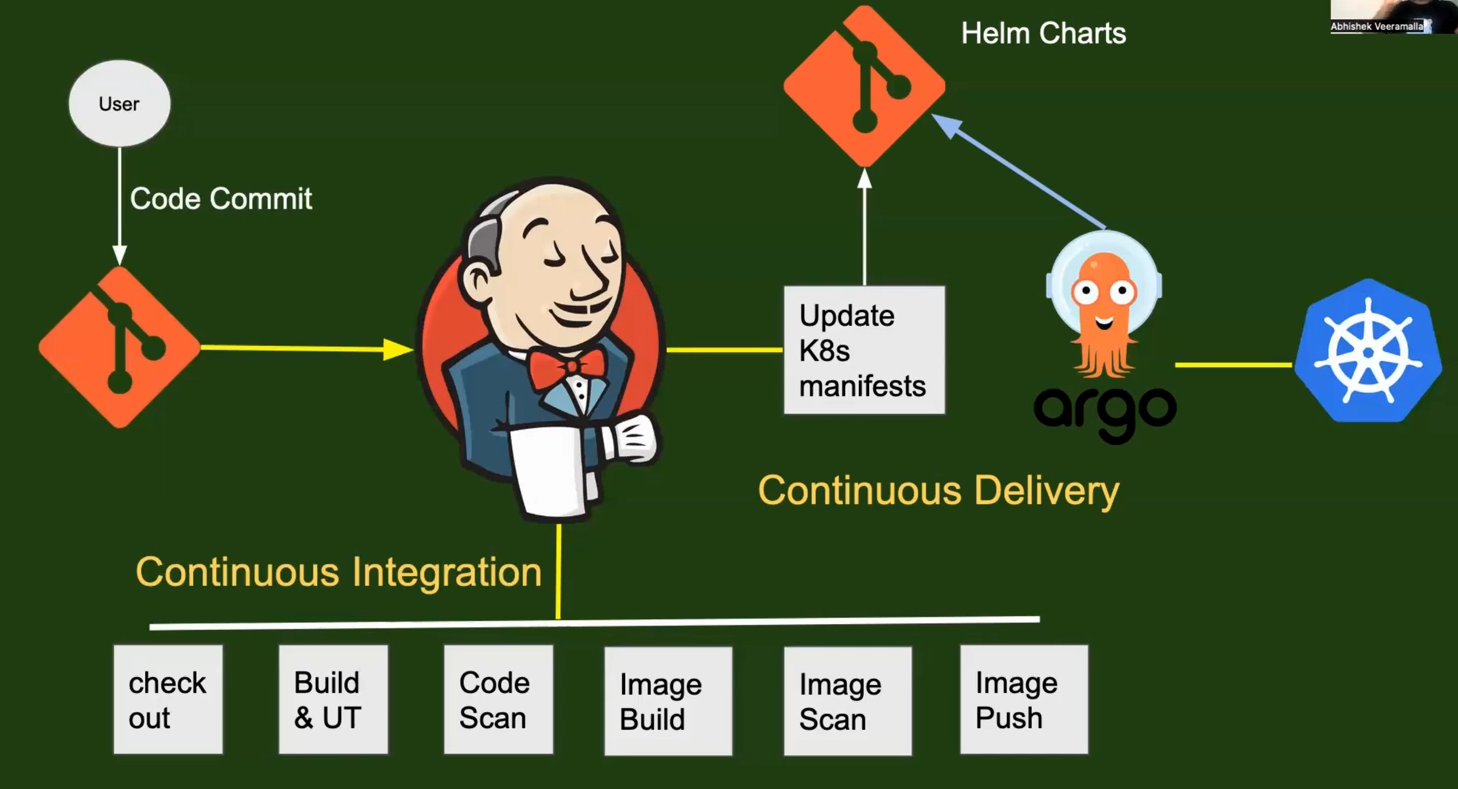
* Stores Code in Private Git Repositories
* **AWS 🡺 Developer Tools 🡺 Code Commit**
* **Terms:** Repository, Push, Pull, Commit, Branches, ADD Files, Version Changes, Cloning, IAM, Code Commit Policies
* Works Like GitHub / Git / GitLab
* Version Control System
* You have to use IAM Role to work with SSH Configurations.
* With UI, you can only upload or edit only 1 file at a time.
* Need to use Git Terminal for uploading multiple files at once.
* You can Clone Repo using HTPPS Clone Link and Git in Local System.
* You can do all the thing which you did in Github here too, almost Same.
* **Advantages :** Managed Git, Scalability, Reliability
* **Disadvantages:** Less Features, AWS Restricted, Less Integrations Outside AWS

**2.AWS Code Pipeline**

* Release Software using Continuous Delivery
* **AWS 🡺 Developer Tools 🡺 Code Pipeline**
* **Terms:** Code Commit, Build, Test, Deploy, Docker, Kubernetes, Images, Cluster, Ec2, Push, CI, Continuous Delivery
* This tools Only works with AWS Services.
* This Is a Pay as you Go Model where Jenkins is free and Open Source.
* Everything is managed by AWS Only.



**Figure 3 : CICD using AWS Tools**



**Figure 4 : CICD using Jenkins and Github**

**3.AWS Code Build**

* Build and Test Code
* **AWS 🡺 Developer Tools 🡺 Code Build**
* **Terms:** Code Commit, Build, Test, Deploy, Docker, Kubernetes, Images, Cluster, Ec2, Push, CI, Continuous Delivery

**Systems Manager**

* AWS Systems Manager is a Central Place to View and Manage AWS.
* **Terms:** Parameters, Store, Secrets, Vault, Passwords
* **AWS 🡺 Management Tools 🡺 Systems Manager**

**AWS Notes**

* 1 – Tier – Only Front End
* 2 – Tier – Front End & Back End
* 3 – Tier – Front End, Back End & Database

**DevOps**

**Concepts: Fundamentals, SDLC, Virtual Machines, AWS, EC2, AWS CLI, Linux, Shell Scripting, Git, GitHub, IAC, Terraform, Ansible, CICD, Docker, JIRA, Kubernetes, Jenkins, Maven**