# CloudIAM - Implementation - Quick setup guide - marketplace

## Pre-requirements

Before starting implementation, you need first:

1. AWS CLI installed on local machine. [INSTRUCTION HERE]
2. AWS Account with Administrator access for CloudIAM product deployment + AWS profile configured on local machine. [INSTRUCTION HERE]
3. The same AWS Account needs at least one AWS VPC + Subnet - needed to build and upload AMI.
4. At least one resource from AWS(EC2/EKS/RDS) is/are available on another AWS account with another VPC - we will be connecting to it from CloudIAM Application as a "Resource".

---

5. Download **infra-cloudiam.zip** file from the Amazon Marketplace product version template and extract it to a separate folder. This will be our ROOT_DIR reference path.
6. Downloaded ECR Images subscribed from Marketplace. Please follow strictly Marketplace instructions to launch images.
7. Please make sure that on local machine are installed:
   a. Python 3.11 or newer,
   b. Terraform 1.9.0 or newer,
   c. Hashicorp Packer
8. Prepared S3 Bucket (for Terraform State) and DynamoDB Table with LockID as a Partition Key(for Terraform Locks). We will use those S3 Bucket and DynamoDB table in already prepared terraform solution. [INSTRUCTION HERE]

## Step 1 - Build required AMIs

Please remember to start working on root directory (top level after unpacking infra-cloudiam.zip)

1. Open extracted "infra-cloudiam.zip" folder as new workspace with editor of choice (for example Visual Studio Code).
2. Open new terminal window in extracted folder.
3. Create and activate new python3 virtual environment

```
python3 -m venv ~/ansible-env
source ~/ansible-env/bin/activate
```

4. Install required ansible packages:

```
pip install ansible
ansible-galaxy collection install community.windows
```

5. Install WinRm package:

```
pip install ansible pywinrm
```

---

### Step 1.1 Build Bastions AMI

1. Navigate to the bastion directory: `cd prereqs/02_images_preparation/bastion_ami`
2. Copy content of `packer` / `variables.sample.pkrvars.hcl` file to. `packer` / `variables.auto.pkrvars.hcl` and **change all "SET_ME" values** to the ones you desire.

```
mv packer/variables.sample.pkrvars.hcl packer/variables.auto.pkrvars.hcl
nano packer/variables.auto.pkrvars.hcl
```

3. Also change values of keys:
   a. "vpc_name",
   b. "subnet_name",
   c. "profile_name",
   at the end of the file (VPC, subnet and profile name should be the ones prepared from requirements, that are in AWS account where CloudIAM application will be deployed).

Example modified content

```
env    = "prd"
```

```
client = "demo"

default_tags = {
  application      = "CloudIAM"
  cost_center_code = "CloudIAM"
  creator          = "john.kovalsky@example.com"
  owner            = "john.kovalsky@example.com"
  environment      = "prd"
  managed_by       = "packer"
}

amazon_linux_2023 = {
  path_to_ansible_playbook = "./ansible/ansible_playbook.yml"
  region                   = "eu-west-1"
  instance_type            = "t2.micro"
  ssh_username             = "ec2-user"
  host_name                = "amazon_linux_2023"
  image_name               = "cloudiam-bastion-amazon-linux-2023"
  version                  = "1.1.0"
  associate_public_ip      = true
  skip_ami_creation        = false
  source_ami_config = {
    name                = "al2023-ami-2023.*-kernel-6.1-x86_64"
    virtualization_type = "hvm"
    root_device_type    = "ebs"
    architecture        = "x86_64"
    owners              = ["amazon"]
    most_recent         = true
  }
  # vpc where instance for ami creation should be launched
  vpc_name     = "cloudiam-app-prd_vpc"
  # subnet where instance for ami creation should be launched
  subnet_name  = "cloudiam-app-prd_pub_sub_eu-west-1a"
  # AWS Profile Name
  profile_name = "a-demo-prd"
}
```

4. Build Bastion AMI(variables will be injected automatically from `variables.auto.pkrvars.hcl` ) :

```
cd packer/
packer init .
packer build .
```

MacOS sometimes raises a problem with threads, to prevent it we need to run the command before packer build: `export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES`

Because of the problems with python and winrm on MacOS, you should also set the environment variable: `export no_proxy="*"`

5. Copy and save Packer build output (**need for STEP 4**). Example:

```
==> Builds finished. The artifacts of successful builds are:
--> linux.amazon-ebs.linux: AMIs were created:
eu-west-1: ami-0d1cd417d0429aa1a
```

---

### Step 1.2 Build Graphical AMI

Please remember to start working on root directory (top level after unpacking infra-cloudiam.zip)

1. Navigate to the graphical directory: `cd prereqs/02_images_preparation/graphical_ami`
2. Rename folder `aws-profile-name` with the name of AWS profile where CloudIAM will be deployed (see Pre-Requirements#2). Example:

```
mv parameters/aws-profile-name parameters/a-demo-prd
```

3. Choose one of json files in `parameters/aws-profile-name` folder and modify it - **change all “SET ME” values** to the desired ones ( `vpc_id` , `subnet_id` and `aws_profile` should be the ones prepared from requirements). Example modified content for `windows-server-2022-base-english.json` :

Example modified content

```
{
  "building": {
    "image_name": "cloudiam-vdi-windows-server-2022-base-english",
    "source_image_name": "Windows_Server-2022-English-Full-Base-2024.10.09",
    "instance_type": "c5.2xlarge",
    "profile_name": "a-demo-prd",
    "region": "eu-west-1"
  },
  "network": {
    "vpc_id": "vpc-0a1234bc56789000d",
    "subnet_id": "subnet-0e5f67g8h901ijk23",
    "associate_public_ip_address": true
  },
  "default_tags": {
    "application": "CloudIAM",
    "cost_center_code": "CloudIAM",
    "creator": "john.kovalsky@example.com",
    "owner": "john.kovalsky@example.com",
    "environment": "prd",
    "managed_by": "packer"
  }
}
```

JSON file in name contain *-english.json or *-polish.json. It reflects to the operating system Language Pack used for Windows Image.

4. Build Graphical AMI image:

```
packer init .
packer build --var-file "./parameters/<aws-profile-name>/windows-server-2022-base-english.json" .
```

Please take a note that building of Windows AMI can take a while …

5. Copy and save packer build output (**we will need it in STEP 4**). Example:

```
==> Builds finished. The artifacts of successful builds are:
--> linux.amazon-ebs.linux: AMIs were created:
eu-west-1: ami-0d1cd417d0429aa1a
```

---

### Step 1.3 Build Guacamole Gateway AMI

Please remember to start working on root directory (top level after unpacking infra-cloudiam.zip

1. Navigate to the directory: `cd prereqs/02_images_preparation/guacamole_gateway_ami`
2. Rename folder `aws-profile-name` with the name of AWS profile where CloudIAM will be deployed (see Pre-Requirments#2.

```
mv parameters/aws-profile-name parameters/a-demo-prd
```

3. Modify content of `prereqs/02_images_preparation/guacamole_gateway_ami/parameters/amazon-linux-2023.json` file - **change all "SET ME" values** to the desired ones (`vpc_id`, `subnet_id` and `aws_profile` should be the ones prepared from requirements). Example modified content for `amazon-linux-2023.json`:

Example modified content

```
{
  "building": {
    "image_name": "cloudiam-vdi-guacamole-gw",
    "source_image_name": "al2023-ami-2023.6.20241121.0-kernel-6.1-x86_64",
    "instance_type": "t3.large",
    "profile_name": "a-demo-prd",
    "region": "eu-west-1"
  },
  "network": {
    "vpc_id": "vpc-0a1234bc56789000d",
    "subnet_id": "subnet-0e5f67g8h901ijk23",
    "associate_public_ip_address": true
  },
  "default_tags": {
    "application": "CloudIAM",
    "cost_center_code": "CloudIAM",
    "creator": "john.kovalsky@example.com",
    "owner": "john.kovalsky@example.com",
    "environment": "prd",
    "managed_by": "packer"
  }
}
```

4. Build Guacamole Gateway AMI image:

```
packer init .
packer build --var-file "./parameters/<aws-profile-name>/amazon-linux-2023.json" .
```

This product includes Apache Guacamole, an open-source software licensed under the Apache License 2.0.

Guacamole's license can be found at:

https://www.apache.org/licenses/LICENSE-2.0

For more information about Apache Guacamole, visit: https://guacamole.apache.org

Please take a note that building of Windows AMI can take a while.

5. Copy and save packer build output (**we will need it in STEP 4**). You can mark it in your notes as "". Example:

```
==> Builds finished. The artifacts of successful builds are:
--> linux.amazon-ebs.linux: AMIs were created:
eu-west-1: ami-0d1cd417d0429aa1a
```

## Step 2 - Enabling cross-account access

Please remember to start working on root directory (top level after unpacking infra-cloudiam.zip

1. Navigate to the directory: `cd prereqs/00_cf_cross_account_roles`
2. Install python packages from `requirements.txt`. Make sure you have already installed and activated Python Env(see PreRequirements#7)

```
python3 -m venv ~/cross-account-env
source ~/cross-account-env/bin/activate
pip install -r requirements.txt
```

3. Fill the configuration in `config.sample.yml` (you can rename this file if needed). If renamed, please remember to adjust command in point 4:

Please **DO NOT MODIFY** content of main.yml or target.yml files

Example modified content

```
main_template:
  stack_name: "prereq-stack-main"
  main_role_name: "main-cloudiam-execution-role"
  target_role_name: "target-cloudiam-execution-role"
  path_to_file: ./main.yml
target_template:
  stack_name: "prereq-stack-target"
  target_role_name: "target-cloudiam-execution-role"
  path_to_file: ./target.yml
deploy_to:
  - account_id: 123456789012
    profile_name: a-demo-prd
    main: true
  - account_id: 234567890123
    profile_name: a-demo-prd-app1
  - account_id: 345678901234
    profile_name: a-demo-dev-app1
```

4. Deploy cross account roles by running python file:

```
python3 manage.py --config-file-path ./config.sample.yml --action deploy
```

**Valid actions**:

- deploy
- update
- destroy

5. From script output **copy and save last part that we will use in terraform configuration in next steps**. Example output to copy:

```
{
    "MainCFIAMExecutionRole": "arn:aws:iam::123456789012:role/main-cloudiam-execution-role",
    "TargetCFIAMRoleArn": "target-cloudiam-execution-role"
}
```

## Step 3 - Setting up R53 and ECR repositories

Please remember to start working on root directory (top level after unpacking infra-cloudiam.zip)

ECR repositories are needed to store CloudIAM images, which will be delivered with the Marketplace product. It's necessary to prepare 3 ECR Repositories:

- BE CloudIAM
- FE CloudIAM
- Lambdas CloudIAM

ECR creation is mandatory to store the images from the marketplace in the remote ECR.

1. Go to `prereqs/01_prereqs` and fill `backend.tf` and `providers.tf` configuration files. You should use there S3 bucket, DynamoDB Table and AWS profile created for this guide's Pre-requirements. Example:

```
"""providers.tf"""

provider "aws" {
  region  = "eu-west-1"
  profile = "a-demo-prd"
}
```

```
"""backend.tf"""

terraform {
  backend "s3" {
    profile        = "a-demo-prd"
    bucket         = "a-demo-prd-s3-cloudiam-tfstate"
    key            = "cloudiam-ecr-r53"
    region         = "eu-west-1"
    dynamodb_table = "a-demo-prd-ddb-tf-state-lock"
  }
}
```

Region in providers.tf and backend.tf must be equal

2. In `prereqs/01_prereqs/terraform.sample.tfvars` fill ecr block values for `backend_repository`, `frontend_repository` and `lambdas_repository` keys and save as a `terraform.auto.tfvars`. Example:

```
ecr = {
  create       = true
  repositories = {
    backend_repository = "a-demo-prd-cloudiam-backend"
    frontend_repository = "a-demo-prd-cloudiam-frontend"
    iac_lambdas_repository = "a-demo-prd-cloudiam-iac-lambdas"
  }
}

r53 = {
  create           = true
  hosted_zone_name = "prd.examplehostedzone.com"
}

# idp_secret used only for advanced integration with identity providers.
# "create" set to false to skip it.
idp_secret = {
  # Setting to false prevent resources creation
  create                  = false
  # Change only tam (for client name shortcut and dev for type of env.
  # Make it aligned with values in main deployment vars.
  name                    = "a-demo-prd-sm-idp-metadata-config"
  recovery_window_in_days = 0
  path_to_files = {
    google = "./google.xml"
  }
}

```

3. In `prereqs/01_prereqs/terraform.auto.tfvars` fill R53 block value of `hosted_zone_name`

- If you already have an existing public hosted zone on the main account, you must pass the **hosted zone id in STEP 4** to the terraform configuration using "**hosted_zone_id**" variable and set "**create**" key to **false** in **R53** block in `prereqs/01_prereqs/terraform.auto.tfvars`.
- If the main CloudIAM account does not have any hosted zones but you have a DNS account in your organization (or another account used for domain registration that can handle public DNS resolution), you must create a hosted zone and, if necessary, delegate name resolution to the main hosted zone in your organization by passing the appropriate NS records.

4. Go to `prereqs/01_prereqs` and create terraform workspace, with the same name as configured AWS Profile for account where CloudIAM will be deployed. Example:

```
terraform init
terraform workspace new a-demo-prd
```

You can list all available workspaces with command:

```
terraform workspace list
```

"*" next to workspace name points to currently selected workspace.

You can also show selected workspace with command:

```
terraform workspace show
```

5. Apply and confirm terraform deployment:

```
terraform apply
```

You can verify resources created by terraform stack before you confirm deployment on `apply` action. If all resources seems to be fine, then proceed with accepting changing, which will trigger deployment.

6. Now please follow Marketplace instructions to push subscribed image to ECR Repositories:
   a. BE image into BE CloudIAM
   b. FE image into FE CloudIAM
   c. Lambdas image into Lambdas CloudIAM

Use output from `terraform apply` command from previous point to get repositories names/uris necessary for pushing images from AWS Marketplace.

## Step 4 - Terraform deployment

Please remember to start working on root directory (top level after unpacking infra-cloudiam.zip)

1. Copy and rename the `sample` folder to match the name of the configured AWS Profile for the account where CloudIAM will be deployed. Example:

```
cp -R "conf/sample" "conf/a-demo-prd"
```

Please be aware that terraform workspace name must equal directory name in conf/ so if you type `terraform workspace new a-demo-stg` then modify above command to `cp -R "conf/sample" "conf/a-demo-stg"`

2. Go to copied folder `cd conf/` and modify files `accounts-to-enroll.yml`, `group-mapping.yml` and `tf-params.yml`. Those file defines accounts, groups and user.

Examples:

```
"""accounts-to-enroll"""

target_session_duration: 3600
accounts_to_enroll:
  eu-west-1:
    - account_aws_id: 123456789012
      # You set your own name, but follow naming convention
      name: a-demo-prd-eu-west-1
      environment: prd
      description: "CloudIAM main account (where CloudIAM is deployed) Region: eu-west-1"
    - account_aws_id: 234567890123
      name: a-demo-prd-app1-eu-west-1
      environment: prd
      description: "Prd account of app1. Region: eu-west-1"
    - account_aws_id: 345678901234
      name: a-demo-dev-app1-eu-west-1
      environment: dev
      description: "Dev account of app1. Region: eu-west-1"
  eu-central-1:
    - account_aws_id: 234567890123
      name: a-demo-prd-app1-eu-central-1
      environment: prd
      description: "Prd account of app1. Region: eu-central-1"
```

If your accounts use other regions, then you can change or add other regions, apart from ones showed in example.

---

```
"""group-mapping.yml"""

users_to_create:
  - username: superuser
    generate_password: true
    email: superuser@example.com
  - username: test
    generate_password: true
    email: test@example.com
  - username: johnkovalsky
    generate_password: true
    email: john.kovalsky@example.com

groups_to_create:
  COGNITO:
    admin:
      admin_group: true
      description: "This is an admin group"
    tmp_users:
      description: "This is a tmp_users group"
    basic_access:
      description: "This is additional group with perms to basic applications"

attach_users_to_groups:
  admin:
    - superuser
    - johnkovalsky
  tmp_users:
    - test
  basic_access:
    - johnkovalsky
    - test
```

While replicating groups to cognito it’s mandatory to create ADMIN and TMP_USERS groups in `group-mapping.yml` file. ADMIN is a main admin group with privileged access to the application. The Groups replication is very important step in GBAC Access Control on API side.

Users MUST belong to either `admin` or `tmp_users` group. Belonging to `admin` group gives access to CloudIAM application administrator panel.

---

In tf-params.yml we need to use data prepared from previous steps:

- `image_id` in `vdi_gateway` block fill with value from STEP 1.3 (output from built guacamole_gateway_ami).
- Use outputs from STEP 2 to fill fields `stack_set_execution_role_arn` ( `MainCFIAMExecutionRole` from STEP 2) and `target_cf_execution_role` ( `TargetCFIAMRoleArn` from STEP 2)
- `ecr_repo_config` block - use repositories names created in STEP 3.

- `allowed_alb_ips` - stores IPs allowed to connect to application. Add there IP addresses of people allowed to use CloudIAM application. This attribute works only if `app_access_external` is set to `true`, otherwise application allows connection only through VPN set up by client (VPN is not a part of application itself).

```yaml
"""tf-params.yml"""
# ---------- General ----------
env: prd
aws_region: eu-west-1
aws_profile: a-demo-prd
az_count: 2
client: demo
service_name: cloudiam-app
domain_name: # set me from pre req 01 (the same as in STEP 3)
route53_zone_id: # set me from pre req 01 (from STEP 3)

# ---------- TEST ---------------
app_access_external: true
allowed_alb_ips:
  - description: "John Kovalsky"
    ip: 88.123.123.12/32
  - description: "Manager"
    ip: 87.321.123.12/32


application:
  name: CloudIAM
  owner: "John Kovalsky Team"
  creator: john.kovalsky@example.com
  cost_center: "Devops"

# This role should be used to create stack sets and deploy on target accounts
stack_set_execution_role_arn: "arn:aws:iam::123456789012:role/main-cloudiam-execution-role"
target_cf_execution_role: "target-cloudiam-execution-role"


# ---------- Networking ----------
vpc_global_cidr_block: 10.21.0.0/16

# Rate Limiter: If waf_enable is set to true, AWS WAF will block requests originating from the same IP address
# after exceeding the request_limit of 6000 requests within a rolling 5-minute window.
# request_limit must be between 10 and 20 0000 000.
waf_enable: true
request_limit: 6000

# ---------- ECS ----------
ecs_task:
  fe:
    vcpu: 2048
    memory: 4096
    container_port: 80
    min_capacity: 1
    desired_count: 1
    max_capacity: 5
  be:
    vcpu: 2048
    memory: 4096
    container_port: 8000
    min_capacity: 1
    desired_count: 1
    max_capacity: 10

# ---------- VDI ----------
vdi_gateway:
  image_id: "ami-0d1cd417d0429aa1a"
  asg:
    min_size: 1
    max_size: 2
    desired_capacity: 1
    instance_type: t3.small

streaming_monitoring_interval_seconds: 30
```

```yaml

# ---------- ECR ----------
ecr_repo_config:
  fe:
    repository_name: "a-demo-prd-cloudiam-frontend"  # <set me from prereq 01>
    image_tag: 250210  # ex: 250210 Tag Version from subscribed Marketplace Product Image
  be:
    repository_name: "a-demo-prd-cloudiam-backend" # <set me from prereq 01>
    image_tag: 250210  # ex: 250210 Tag Version from subscribed Marketplace Product Image
  lambdas:
    repository_name: "a-demo-prd-cloudiam-iac-lambdas"  # <set me from prereq 01>
    image_tag: 250210  # ex: 250210 Tag Version from subscribed Marketplace Product Image

disable_time_in_seconds: 60
max_allowed_login_attempts: 5

# ---------- AI Analysis ----------
feature_log_analysis: true
feature_log_analysis_model_region: us-east-1
feature_log_analysis_model_id: "anthropic.claude-3-5-sonnet-20240620-v1:0"
feature_log_analysis_model_name: "Claude 3.5 Sonnet"
feature_log_analysis_model_provider: "Anthropic"

```

3. Go to `infra-cloudiam/conf/<env_name>/iac` (env_name is name of the folder you set in point 2) and modify all files, according to your needs. These configuration files are responsible for loading already existing data into CloudIAM application, for example adding EC2 resource makes it possible to use it in application and connect to it through CloudIAM bastion.

Examples:

```yaml
"""ami-mapping.yml"""

cloudiam-vdi-windows-server-2022-base-english-20250131123456: ami-0d1cd417d0429aa1a
cloudiam-bastion-amazon-linux-2023-1.1.0: ami-0d1cd417d0429aa1a
```

---

```yaml
"""application-mapping.yml"""

# Allows generating access to specified AWS account with permissions defined by policies
console_application_1:
  description: First console application
  duration: 3600
  app_type: CONSOLE
  policies: [ "a-demo-dev-app1-admin-access" ]
  groups: [ "admin" ]
  users: [ "test" ]
  account: a-demo-dev-app1-eu-west-1

bastion_application_1:
  description: First bastion application with access to account a-demo-dev-app1-eu-west-1
  duration: 4000
  instance_type: t3.nano
  app_type: BASTION
  cidrs: [ "0.0.0.0/0" ]
  policies: [ "a-demo-dev-app1-admin-access" ]
  groups: [ "admin", "basic_access" ]
  users: []
  account: a-demo-dev-app1-eu-west-1
  ami: cloudiam-bastion-amazon-linux-2023-1.1.0
  aws_resources: [ "test_ec2", "test_rds", "test_eks" ]

graphical_application_1:
  description: First VDI application
  duration: 3600
  instance_type: t3.xlarge
  app_type: GRAPHICAL
  cidrs: [ "0.0.0.0/0", "192.168.8.1/32" ]
  policies: [ "a-demo-dev-app1-admin-access" ]
  groups: []
  users: [ "johnkovalsky" ]
  account: a-demo-dev-app1-eu-west-1
  ami: cloudiam-vdi-windows-server-2022-base-english-20250131123456
  aws_resources: [ "test_ec2", "test_rds", "test_eks" ]

```

---

```yaml
"""aws-policy-resources.yml"""
# Policies are used for granting permissions in defined applications

a-demo-prd-admin-access:
  aws_managed: true
  account: a-demo-prd-eu-west-1
  description: "This is an administrator access for prd account"
  policy_arn: arn:aws:iam::aws:policy/AdministratorAccess

a-demo-prd-app1-ddb-access:
  aws_managed: true
  account: a-demo-prd-app1-eu-west-1
  description: "This is a full access to DynamoDB service"
  policy_arn: arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess

a-demo-dev-app1-admin-access:
  aws_managed: true
  account: a-demo-dev-app1-eu-west-1
  description: "This is an administrator access for dev app1 account"
  policy_arn: arn:aws:iam::aws:policy/AdministratorAccess

a-demo-prd-app1-ro-access:
  aws_managed: true
  account: a-demo-prd-app1-eu-west-1
  description: "This is read only access to DynamoDB service"
  policy_arn: arn:aws:iam::aws:policy/AmazonDynamoDBReadOnlyAccess
```

---

```yaml
"""aws-ec2-resources.yml"""

EC2:
  test_ec2:
    description: EC2 instance on a-demo-dev-app1-eu-west-1 account
    additional_config:
      security_group_ids: [ "sg-0ccadd6543b123ab1"]
    account: a-demo-dev-app1-eu-west-1
```

---

```yaml
"""aws-eks-resources.yml"""

test-eks:
  description: eks with access to cluster a-demo-dev-app1-eks
  additional_config:
    security_group_ids: ["sg-0e3fcab21bb192j38", "sg-0b79d5bf51a123456"]
    eks_access_config:
      a-demo-dev-app1-eks:
        AmazonEKSClusterAdminPolicy:
          cluster: False
          namespaces: [ "default"]
        AmazonEKSAdminViewPolicy:
          cluster: True
          namespaces: []
  account: a-demo-dev-app1-eu-west-1
```

---

```yaml
"""aws-rds-resources.yml"""

test-rds:
  description: a-demo-dev-app1 rds resource
  additional_config:
    security_group_ids: ["sg-042de95c4d618dj27"]
  account: a-demo-dev-app1-eu-west-1

```

---

4. In main folder `infra-cloudiam` fill `backend.tf` configuration file. You should use there S3 bucket, DynamoDB Table and AWS profile created for this guide's Pre-requirements. Example:

```
terraform {
  backend "s3" {
    profile        = "a-demo-prd"
    bucket         = "a-demo-prd-s3-cloudiam-tfstate"
    key            = "cloudiam-main-app"
    region         = "eu-west-1"
    dynamodb_table = "a-demo-prd-ddb-tf-state-lock"
  }
}
```

5. Go to main folder `infra-cloudiam` and create terraform workspace, with the same name as configured AWS Profile for account where CloudIAM will be deployed. Example:

```
terraform init
terraform workspace new a-demo-prd
```

You can list all available workspaces with command:

```
terraform workspace list
```

“*” next to workspace name points to currently selected workspace.

You can also show selected workspace with command:

```
terraform workspace show
```

Workspaces created now are separate from workspaces made in STEP 3.

6. Deploy with commands (you need to have your new workspace selected before running command):

```
terraform plan
terraform apply
```

You can verify resources created by terraform stack before you confirm deployment on `apply` action. If all resources seems to be fine, then proceed with accepting changing, which will trigger deployment.

## Appendix

### AWS Resources Naming Convention and Limitations

[EXPAND ME] Quotas

| Description | Limit |
|---|---|
| Alias for an AWS account ID | 3–63 characters |
| For inline policies | You can add as many inline policies as you want to an IAM user, role, or group. But the total aggregate policy size (the sum size of all inline policies) per entity can't exceed the following limits:<br>• User policy size can't exceed 2,048 characters.<br>• Role policy size can't exceed 10,240 characters.<br>• Group policy size can't exceed 5,120 characters.<br>Note<br>IAM doesn't count white space when calculating the size of a policy against these limits. |
| For managed policies | • The size of each managed policy can't exceed 6,144 characters.<br>Note<br>IAM doesn't count white space when calculating the size of a policy against this limit. |
| Group name | 128 characters |
| Instance profile name | 128 characters |
| Password for a login profile | 1–128 characters |
| Path | 512 characters |
| Policy description | After creation - immutable field. In order to change the description the policy must be recreated. |
| Policy name | 128 characters |
| Role name | 64 characters<br>Important<br>If you intend to use a role with the **Switch Role** feature in the AWS Management Console, then the combined `Path` and `RoleName` can't exceed 64 characters. |
| Role session duration | 12 hours<br>When you assume a role from the AWS CLI or API, you can use the `duration-seconds` CLI parameter or the `DurationSeconds` API parameter to request a longer role session. You can specify a value from 900 seconds (15 minutes) up to the maximum session duration setting for the role, which can range 1–12 hours. If you don't specify a value for the `DurationSeconds` parameter, your security credentials are valid for one hour. IAM users who switch roles in the console are granted the maximum session duration, or the remaining time in the user's session, whichever is less. The maximum session duration setting doesn't limit sessions assumed by AWS services. To learn how to view |

| | |
|---|---|
| | the maximum value for your role, see View the maximum session duration setting for a role. |
| Role session name | 64 characters |
| Role session policies | • The size of the passed JSON policy document and all passed managed policy ARN characters combined can't exceed 2,048 characters.<br>• You can pass a maximum of 10 managed policy ARNs when you create a session.<br>• You can pass only one JSON policy document when you programmatically create a temporary session for a role or federated user.<br>• Additionally, an AWS conversion compresses the passed session policies and session tags into a packed binary format that has **a separate limit**. The `PackedPolicySize` response element indicates by percentage how close the policies and tags for your request are to the upper size limit.<br>• We recommend that you pass session policies using the AWS CLI or AWS API. The AWS Management Console might add additional console session information to the packed policy. |
| Role session tags | • Session tags must meet the tag key limit of 128 characters and the tag value limit of 256 characters.<br>• You can pass up to 50 session tags.<br>• An AWS conversion compresses the passed session policies and session tags into a packed binary format that has a separate limit. You can pass session tags using the AWS CLI or AWS API. The `PackedPolicySize` response element indicates by percentage how close the policies and tags for your request are to the upper size limit. |
| SAML authentication response base64 encoded | 100,000 characters<br><br>This character limit applies to `assume-role-with-saml` CLI or `AssumeRoleWithSAML` API operation. |
| Tag key | 128 characters<br><br>This character limit applies to tags on IAM resources and session tags. |
| Tag value | 256 characters<br><br>This character limit applies to tags on IAM resources and session tags.<br><br>Tag values can be empty which means tag values can have a length of 0 characters. |
| Unique IDs created by IAM | 128 characters. For example:<br>• User IDs that begin with `AIDA`<br>• Group IDs that begin with `AGPA`<br>• Role IDs that begin with `AROA`<br>• Managed policy IDs that begin with `ANPA`<br>• Server certificate IDs that begin with `ASCA`<br><br>Note<br><br>This isn't intended to be an exhaustive list, nor is it a guarantee that IDs of a certain type begin only with the specified letter combination. |
| User name | 64 characters |

### Tagging

All resources created by Terraform are tagged using common "default" tags that are application-specific tags and customer tags that allow customers to tag resources with tags matching their tagging convention.

Application-specific tags should never be changed.

Customer tags can be defined in a file in the following location: **conf/customer_tags.yaml**

[EXPAND ME] Example customer tags definition

```yaml
customer_tags:
  description: "Customer tags to apply to CloudIAM resources"
  type: map
  default: {}
  validation:
    condition: "!contains(keys(customer_tags), 'Name')"
    error_message: "The customer_tags variable must not contain a tag with the key 'Name'."
  tags:
    cost_center_code: "Devops"
    maintainer: "John Kovalsky Team"
    owner: "John Kovalsky Team"

```