SSM Document YAML (TKT 59) – Break Glass:

---

schemaVersion: "2.2"

description: "Command to create user using ssm document"

parameters:

UserName:

type: "String"

description: "Create user - break-glass"

default: "break-glass"

mainSteps:

- action: "aws:runShellScript"

name: "CreateUser"

inputs:

runCommand:

- sudo useradd break-glass

- cd /home/break-glass

- sudo mkdir .ssh

- sudo chmod 700 .ssh

- sudo echo -e "ssh-rsa  santiagorivera@MBP-M1.local" >> .ssh/authorized\_keys

- sudo chmod 600 .ssh/authorized\_keys

Splunk:

Warning: splunkforwarder-9.0.3-dd0128b1f8cd-linux-2.6-x86\_64.rpm: Header V4 RSA/SHA256 Signature, key ID b3cd4420: NOKEY

Complete

Checkmk – Ubuntu

Created new site ytmonitoring with version 2.1.0p20.cre.

The site can be started with omd start ytmonitoring.

The default web UI is available at http://ip-172-31-54-49/ytmonitoring/

The admin user for the web applications is cmkadmin with password: YuCwdpbM

For command line administration of the site, log in with 'omd su ytmonitoring'.

After logging in, you can change the password for cmkadmin with 'cmk-passwd cmkadmin'.

oot@ip-172-31-54-49:~# omd start pythonmonitoring

omd: The site 'pythonmonitoring' does not exist. You need to execute omd as root or site user.

root@ip-172-31-54-49:~# omd version

OMD - Open Monitoring Distribution Version 2.1.0p20.cre

root@ip-172-31-54-49:~# omd create pythonmonitoring

Adding /opt/omd/sites/pythonmonitoring/tmp to /etc/fstab.

Creating temporary filesystem /omd/sites/pythonmonitoring/tmp...OK

Updating core configuration...

Generating configuration for core (type nagios)...

Precompiling host checks...OK

Executing post-create script "01\_create-sample-config.py"...OK

Restarting Apache...OK

Created new site pythonmonitoring with version 2.1.0p20.cre.

The site can be started with omd start pythonmonitoring.

The default web UI is available at http://ip-172-31-54-49/pythonmonitoring/

The admin user for the web applications is cmkadmin with password: hiYpfCTp

For command line administration of the site, log in with 'omd su pythonmonitoring'.

After logging in, you can change the password for cmkadmin with 'cmk-passwd cmkadmin'.

RDP:

~~When prompted, connect to your instance using the following details:~~

~~Public DNS~~

~~ec2-18-207-112-79.compute-1.amazonaws.com~~

~~User name~~

~~Administrator~~

~~Password~~

~~p.mU!P4Q%QWjYvJR;)c&0B%\*KEOLL1?@~~

~~If you've joined your instance to a directory, you can use your directory credentials to connect to your instance.~~

When prompted, connect to your instance using the following details:

User name

 Administrator

Password

 p.mU!P4Q%QWjYvJR;)c&0B%\*KEOLL1?@

[Fleet Manager Remote Desktop](https://us-east-1.console.aws.amazon.com/systems-manager/managed-instances/rdp-connect?region=us-east-1&instances=i-09e29235cfe92ed9f)

If you've joined your instance to a directory, you can use your directory credentials to connect to your instance.

New 09Feb23:

root@ip-172-31-60-94:~# omd version

OMD - Open Monitoring Distribution Version 2.1.0p18.cre

root@ip-172-31-60-94:~# omd create ytmonitoring

Adding /opt/omd/sites/ytmonitoring/tmp to /etc/fstab.

Creating temporary filesystem /omd/sites/ytmonitoring/tmp...OK

Updating core configuration...

Generating configuration for core (type nagios)...

Precompiling host checks...OK

Executing post-create script "01\_create-sample-config.py"...OK

Restarting Apache...OK

Created new site ytmonitoring with version 2.1.0p18.cre.

The site can be started with omd start ytmonitoring.

The default web UI is available at http://ip-172-31-60-94/ytmonitoring/

The admin user for the web applications is cmkadmin with password: fHxsHkY9

For command line administration of the site, log in with 'omd su ytmonitoring'.

After logging in, you can change the password for cmkadmin with 'cmk-passwd cmkadmin'.

RDP:

Connection Type

Connect using RDP clientDownload a file to use with your RDP client and retrieve your password.

Connect using Fleet ManagerConnect to your instance using Fleet Manager Remote Desktop.

When prompted, connect to your instance using the following details:

User name

 Administrator

Password

 4d-?Ft@t2NZF\*ta3s9VHlP5Mt\*pMP20k

Brew installation on mac

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"

Chatgpt:

AWS CloudFormation sample EC2, VPC, InternetGateway, Subnets, Routing table, ASG, Launch Template, Target Group, Security Group, User data install Apache, any region, option to select instance type, option to select region, default instance type t2.micro, default region us-east-1a.

TKT-37 Userdata:

#!/bin/bash

sudo yum -y update

sudo yum install -y httpd

sudo systemctl start httpd

sudo systemctl enable httpd

sudo yum install -y git

sudo yum install ruby wget -y

cd /home/ec2-user

sudo wget https://aws-codedeploy-us-east-1.s3.us-east-1.amazonaws.com/latest/install

sudo chmod +x ./install

sudo ./install auto

sudo git config --system credential.helper '!aws codecommit credential-helper $@'

sudo git config --system credential.UseHttpPath true

sudo git config --system credential.UseHttpsPath true

sudo git config --system user.name "Mathewos"

sudo git config --system user.email "matmar2@yahoo.com"

cd /home/ec2-user

sudo git clone -b main <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/TKT-MAT-37-CodeCommit-Repo> /home/ec2-user/TKT-MAT-37-CodeCommit-Repo

vpn site-to-site connection between your own account and the organization account using openswam

<https://www.youtube.com/watch?v=7tTrN8WXMlg&ab_channel=DigitalCloudTraining>

**Lambda needs a role that will allow it to stop and start EC2 instances in your account. Make sure your Role does not allow Lambda to perform other activities.**

Created policy “Start-Stop-Instance-Policy-TKT-62”

Created role “Start-Stop-Instance-Role-TKT-62”

**Policy:**

{  
"Version": "2012-10-17",  
"Statement": [  
{  
"Sid": "VisualEditor0",  
"Effect": "Allow",  
"Action": [  
"ec2:StartInstances",  
"ec2:StopInstances",  
"logs:PutLogEvents"  
],  
"Resource": [  
"arn:aws:license-manager::202618001640:license-configuration:",  
"arn:aws:ec2::202618001640:instance/",  
"arn:aws:logs::202618001640:log-group::log-stream:"  
]  
},  
{  
"Sid": "VisualEditor1",  
"Effect": "Allow",  
"Action": [  
"logs:CreateLogStream",  
"logs:CreateLogGroup"  
],  
"Resource": "arn:aws:logs::202618001640:log-group:\*"  
}  
]  
}

**Role thrust relationship/entities:**

{  
"Version": "2012-10-17",  
"Statement": [  
{  
"Effect": "Allow",  
"Principal": {  
"Service": "[lambda.amazonaws.com](http://lambda.amazonaws.com)"  
},  
"Action": "sts:AssumeRole"  
}  
]  
}

MAT-59 Requirements:

* Create an SSM document to deploy this user so that it can be used again in the future. In your SSM document include some commands that verify that the user has been added.
* Since the user will not be able to login with a password you will need to add the following public Key to the authorized key file.
* ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDg4Tss2eHWT2Z/6SYYMNb2hzmcl0hoxckGhTf3KVIVpRFrfRks3Qnh2oGycey2icwYfrun1vWm6rmUug9hZTVTHfsabj++xylEu/8XuQto0HeWp9tk4iwNku6EZ+JBzuR7iHrVZcwqSCQn5ln9SaKALO
* uCqoDNQq/gWIYPmZVE0WD+66pIfKuUvaADNpVecn4B+AGsfBgAeKVz63zXqxbR8Y3Hjl2rvVfGBSGj9kxykH9klI0ew3falGh3D5JNpxXRyWg2u9LkLXEqqXRJXF9JqRKT0ZqTLaprWqOe6U7DoHV8ktyYyE6F1WqwjU4g0f8+gYQ8DgQK8ijcwnZmnfAo
* 21RniftymQRoGx6rr3o0TshA61WwC/lI90jnLakt89KWlEvMtb4jWkL0a9NuaUQf21c29lYvQfLSRyqqMXanGrkcrhCG0iY39gwAYVPMotulycd4BEOxv9sVDCoeHcDkJZw4XYVpSJvI1+vh41gquJJLEuA3Sq997uxSrOSjCPncOqSeOd63TJYXYk1hR6
* rbkipoqwglDrRbWKFZo+aWYpBISDV/ap0a9S4c3ltRrwJN6BgI5ZvHWEw0wUQA/VKnRlFtnvNYylIlMFnDjzDhgRxpunvmPpaz1cl1uiM398mNuOyQikxxqZovdkEYveJWssOM/V5ulY9HYGTaRAEO1w== santiagorivera@MBP-M1.local

Here is a link with information on how to accomplish this:

<https://aws.amazon.com/premiumsupport/knowledge-center/new-user-accounts-linux-instance/>

SSM Document:

---

schemaVersion: "2.2"

description: "Command to create user using ssm document"

parameters:

UserName:

type: "String"

description: "Create user - break-glass"

default: "break-glass"

mainSteps:

- action: "aws:runShellScript"

name: "CreateUser"

inputs:

runCommand:

- sudo yum update -y

- sudo useradd break-glass

- sudo su - break-glass

- cd /home/break-glass

- mkdir .ssh

- chmod 700 .ssh

- echo -e "ssh-rsa  santiagorivera@MBP-M1.local" >> .ssh/authorized\_keys

- chmod 600 .ssh/authorized\_keys

# How to install Splunk on a Linux Instance in AWS

<https://www.youtube.com/watch?v=0CHaDfNI4Sg&ab_channel=EmekaakaOn1Productions>

Pritunl Client:

<https://client.pritunl.com/#install>

# AWS EC2 + Autoscaling + Load Balancer + CodeDeploy | Deploy Code At Scale | DevOps With AWS

<https://www.youtube.com/watch?v=Ekgi2HfnJcw&t=1636s&ab_channel=SandipDas>

Github:

HTTPS:

<https://github.com/matmar2/python.git>

Github CLI:

gh repo clone matmar2/python

ChatGPT

<https://openai.com/blog/chatgpt/>

# Setup an AWS Site-to-Site Virtual Private Network (VPN)

<https://www.youtube.com/watch?v=7tTrN8WXMlg>

# AWS VPC Peering Connection Concept with Demo | VPC Peering | AWS VPC Peering Step by Step | AWS Demo

<https://www.youtube.com/watch?v=q-NTKPb16SM>

Terraform for beginners:

<https://kodekloud.com/courses/lab-terraform-for-beginners/>

# ChatGPT Tutorial - Use ChatGPT for DevOps tasks to 10x Your Productivity

<https://www.youtube.com/watch?v=l-kE11fhfaQ>

Checkmk:

<https://checkmk.com/product/features>

# Episode 1: Installing Checkmk and monitoring your first host

<https://www.youtube.com/watch?v=opO-SOgOJ1I&ab_channel=Checkmk>

# How to monitor your Linux servers with Checkmk

<https://www.techrepublic.com/article/how-to-monitor-your-linux-servers-with-checkmk/>

Apprenticeship Site – Procore

<https://sites.google.com/procoreplus.com/apprenticeship-site-aws/tool-guide?authuser=0>

# [ AWS 14 ] Set up your first AWS CodeCommit Repository

<https://www.youtube.com/watch?v=E1GJqfIEJkM&ab_channel=JustmeandOpensource>

### openSUSE, SUSE

<https://packages.cisofy.com/community/#centos-rhel>

# (SAA-C02) AWS Certified Solutions Architect - Associate

<https://www.youtube.com/watch?v=6S0_rNYzcPs&ab_channel=ChienDuong>

Cloudformation and Terraform templates from existing AWS setup - Former2

[www.former2.com](http://www.former2.com)

# Terraform S3 Backend Best Practices

<https://technology.doximity.com/articles/terraform-s3-backend-best-practices>

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 4.16"

}

}

}

# Configure provider

provider "aws" {

region = "us-east-1"

}

# Create VPC

resource "aws\_vpc" "vpc" {

cidr\_block = "10.0.0.0/16"

instance\_tenancy = "default"

tags = {

Name = "prod-VPC"

}

}

# Create internet gateway

resource "aws\_internet\_gateway" "ig" {

vpc\_id = aws\_vpc.vpc.id

tags = {

Name = "ig-project"

}

}

# Create public subnets

resource "aws\_subnet" "public\_1" {

vpc\_id = aws\_vpc.vpc.id

cidr\_block = "10.0.1.0/24"

availability\_zone = "us-east-1a"

map\_public\_ip\_on\_launch = true

tags = {

Name = "public-1"

}

}

# Create route table to internet gateway

resource "aws\_route\_table" "project\_rt" {

vpc\_id = aws\_vpc.vpc.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.ig.id

}

tags = {

Name = "project-rt"

}

}

# Associate public subnets with route table

resource "aws\_route\_table\_association" "public\_route\_1" {

subnet\_id = aws\_subnet.public\_1.id

route\_table\_id = aws\_route\_table.project\_rt.id

}

# Create security groups

resource "aws\_security\_group" "public\_sg" {

name = "public-sg"

description = "Allow web and ssh traffic"

vpc\_id = aws\_vpc.vpc.id

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

# Security group for ALB

resource "aws\_security\_group" "alb\_sg" {

name = "alb-sg"

description = "security group for alb"

vpc\_id = aws\_vpc.vpc.id

ingress {

from\_port = "0"

to\_port = "0"

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = "0"

to\_port = "0"

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

# Create ALB

resource "aws\_lb" "project\_alb" {

name = "alb"

internal = false

load\_balancer\_type = "application"

security\_groups = [aws\_security\_group.alb\_sg.id]

subnets = [aws\_subnet.public\_1.id, aws\_subnet.public\_2.id]

}

# Create ALB target group

resource "aws\_lb\_target\_group" "project\_tg" {

name = "project-tg"

port = 80

protocol = "HTTP"

vpc\_id = aws\_vpc.vpc.id

depends\_on = [aws\_vpc.vpc]

}

# Create target attachments

resource "aws\_lb\_target\_group\_attachment" "tg\_attach1" {

target\_group\_arn = aws\_lb\_target\_group.project\_tg.arn

target\_id = aws\_instance.web1.id

port = 80

depends\_on = [aws\_instance.web1]

}

# Create listener

resource "aws\_lb\_listener" "listener\_lb" {

load\_balancer\_arn = aws\_lb.project\_alb.arn

port = "80"

protocol = "HTTP"

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.project\_tg.arn

}

}

// -----------------------------------------------

// Change USERDATA varible value after grabbing RDS endpoint info

// -----------------------------------------------

data "template\_file" "user\_data" {

template = file("userdata.sh")

vars = {

db\_username = var.database\_user

db\_user\_password = var.database\_password

db\_name = var.database\_name

db\_RDS = aws\_db\_instance.wordpressdb.endpoint

}

}

# Create ec2 instances

resource "aws\_instance" "web1" {

ami = "ami-0cff7528ff583bf9a"

instance\_type = "t2.micro"

key\_name = "Mat-internship-ticket-key"

availability\_zone = "us-east-1a"

vpc\_security\_group\_ids = [aws\_security\_group.public\_sg.id]

subnet\_id = aws\_subnet.public\_1.id

associate\_public\_ip\_address = true

user\_data = <<-EOF

#!/bin/bash

# variable will be populated by terraform template

db\_username=${db\_username}

db\_user\_password=${db\_user\_password}

db\_name=${db\_name}

db\_RDS=${db\_RDS}

# install LAMP Server

yum update -y

#install apache server and mysql client

yum install -y httpd

yum install -y mysql

#first enable php7.xx from amazon-linux-extra and install it

amazon-linux-extras enable php7.4

yum clean metadata

yum install -y php php-{pear,cgi,common,curl,mbstring,gd,mysqlnd,gettext,bcmath,json,xml,fpm,intl,zip,imap,devel}

#install imagick extension

yum -y install gcc ImageMagick ImageMagick-devel ImageMagick-perl

pecl install imagick

chmod 755 /usr/lib64/php/modules/imagick.so

cat <<EOF >>/etc/php.d/20-imagick.ini

extension=imagick

EOF

systemctl restart php-fpm.service

systemctl start httpd

user\_data = data.template\_file.user\_data.rendered

# Change OWNER and permission of directory /var/www

usermod -a -G apache ec2-user

chown -R ec2-user:apache /var/www

find /var/www -type d -exec chmod 2775 {} \;

find /var/www -type f -exec chmod 0664 {} \;

sudo yum -y install mariadb-server

sudo service mariadb start

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Installing Wordpress using WP CLI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar

chmod +x wp-cli.phar

mv wp-cli.phar /usr/local/bin/wp

wp core download --path=/var/www/html --allow-root

wp config create --dbname=$db\_name --dbuser=$db\_username --dbpass=$db\_user\_password --dbhost=$db\_RDS --path=/var/www/html --allow-root --extra-php <<PHP

define( 'FS\_METHOD', 'direct' );

define('WP\_MEMORY\_LIMIT', '128M');

PHP

# Change permission of /var/www/html/

chown -R ec2-user:apache /var/www/html

chmod -R 774 /var/www/html

# enable .htaccess files in Apache config using sed command

sed -i '/<Directory "\/var\/www\/html">/,/<\/Directory>/ s/AllowOverride None/AllowOverride all/' /etc/httpd/conf/httpd.conf

#Make apache autostart and restart apache

systemctl enable httpd.service

systemctl restart httpd.service

echo WordPress Installed

tags = {

Name = "wordpress\_instance"

}

}

# Database subnet group

resource "aws\_db\_subnet\_group" "db\_subnet" {

name = "db-subnet"

subnet\_ids = [aws\_subnet.private\_1.id, aws\_subnet.private\_2.id]

}

# Create database instance

resource "aws\_db\_instance" "project\_db" {

allocated\_storage = 5

engine = "mysql"

engine\_version = "5.7"

instance\_class = "db.t3.micro"

identifier = "db-instance"

db\_name = "project\_db"

username = "admin"

password = "password"

db\_subnet\_group\_name = aws\_db\_subnet\_group.db\_subnet.id

vpc\_security\_group\_ids = [aws\_security\_rds.id]

publicly\_accessible = false

skip\_final\_snapshot = true

}

resource "aws\_security\_group" "rds" {

name = "rds"

description = "Allow RDS traffic from EC2 instances"

ingress {

from\_port = 3306

to\_port = 3306

protocol = "tcp"

security\_groups = [aws\_security\_group.public\_sg.id]

}

}

# Create an S3 bucket for the Terraform state files

resource "aws\_s3\_bucket" "terraform\_state\_bucket" {

bucket = "mat-78-terraform-bucket"

acl = "private"

versioning {

enabled = true

}

}

# Configure the Terraform backend to use S3

terraform {

backend "s3" {

bucket = "mat-78-terraform-bucket"

key = "terraform.tfstate"

region = "us-east-1"

}

}

# Outputs

# Ec2 instance public ipv4 address

output "ec2\_public\_ip" {

value = aws\_instance.web1.public\_ip

}

# Db instance address

output "db\_instance\_address" {

value = aws\_db\_instance.project\_db.address

}

# Getting the DNS of load balancer

output "lb\_dns\_name" {

description = "The DNS name of the load balancer"

value = "${aws\_lb.project\_alb.dns\_name}"

}

MAT-49

#!/bin/bash

# Update the package index

sudo yum update -y

# Install Apache Web Server

sudo yum install -y httpd

# Start the Apache Web Server

sudo service httpd start

# Enable the Apache Web Server to start at boot

sudo chkconfig httpd on

# Display a message

echo "\* \* \* \* \* \* \* W A R N I N G \* \* \* \* \* \* \* \* \* \*  
This computer system is the property of ProCore Plus. It is for authorized use only. By using this system, all users acknowledge notice of, and agree to comply with, the Acceptable Use of Information Technology Resources Policy (AUP).   Unauthorized or improper use of this system may result in administrative disciplinary action, civil charges/criminal penalties, and/or other sanctions as set forth in the AUP. By continuing to use this system you indicate your awareness of and consent to these terms and conditions of use. **LOG OFF IMMEDIATELY** if you do not agree to the conditions stated in this warning.

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" > /var/www/html/index.html

sudo mkdir /mnt/efs

sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-0d6526450707e8392.efs.us-east-1.amazonaws.com:/ /mnt/efs

sudo useradd -c "Mathewos G" mat

#sudo passwd mat (abcd1234)

sudo mkdir -p /mnt/efs/mat

sudo chown mat:mat /mnt/efs/mat

sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-0d6526450707e8392.efs.us-east-1.amazonaws.com:/mat /home/mat

sudo mkdir /mnt/efs/mat/home-directories

# MAT-60: create an Ansible playbook to install the CloudWatch agent on all bastion hosts. The playbook should also put custom metric such as memory and disk space from the bastion host

---

- name: Install CloudWatch agent and send custom metrics on bastion hosts

hosts: local

gather\_facts: false

tasks:

- name: Gather a list of instances with the tag "EC2-TKT-49-54-Bastion Instance"

ec2\_instance\_info:

filters:

"tag:Name": "EC2-TKT-49-54-Bastion Instance"

register: bastion\_instances

- name: Install dependencies

package:

name: awslogs

state: present

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Download the CloudWatch agent configuration file

command: aws s3 cp s3://my-cloudwatch-bucket/cloudwatch-agent-config.json /tmp/cloudwatch-agent-config.json

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Install the CloudWatch agent

command: /usr/local/bin/aws cloudwatch configure-agent --input-path /tmp/cloudwatch-agent-config.json

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Start the CloudWatch agent

service:

name: awslogs

state: started

enabled: yes

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Send custom metrics - Memory Usage

shell: echo "{{ item.private\_ip }} memory\_usage `free -m | grep Mem | awk '{print $3/$2 \* 100.0}'`" | /usr/local/bin/aws cloudwatch put-metric-data --metric-name MemoryUsage --unit Percent --value `free -m | grep Mem | awk '{print $3/$2 \* 100.0}'` --dimensions InstanceId={{ item.instance\_id }}

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Send custom metrics - Disk Space Utilization

shell: echo "{{ item.private\_ip }} disk\_utilization `df -h / | grep / | awk '{print $5}' | sed 's/%//g'`" | /usr/local/bin/aws cloudwatch put-metric-data --metric-name DiskSpaceUtilization --unit Percent --value `df -h / | grep / | awk '{print $5}' | sed 's/%//g'` --dimensions InstanceId={{ item.instance\_id }}

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

Boto3 documentation – Available Services

<https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/index.html>

SSH from Bastion server to Private instance:

ssh-add -K privatekey.pem    (-K for Mac -> agent forwarding -> keypair to be available for the private instance)  
ssh -A ec2-user@public-ipin Bastion host  
ssh ec2-user@private-ip

ssh-add -c privatekey.pem (for Windows)

ssh -A userid@remoteip -> -A used to forward the ssh-agent to the remote host

# Full Stack DevOps Environment Setup Bash Script for Ubuntu, CentOS and MacOS

<https://awstip.com/full-stack-devops-environment-setup-bash-script-for-ubuntu-and-macos-44cd4b68a34>

# ****Installation of LAMP stack on Amazon Linux OS on AWS Ec2 Instance****

<https://awstip.com/installation-of-lamp-stack-on-amazon-linux-os-on-aws-ec2-instance-741114892bea>

Copy from EC2 to S3 bucket – “s3-ticket-mat-59” (then download from S3)

sudo aws s3 cp main.tf s3://s3-ticket-mat-59/main-18Feb23.tf

Copy a directory from local directory to EC2 instance

sudo scp -r -i Mat-internship-ticket-key.pem /Users/matmar2/Downloads/ariclaw-master/ [ec2-user@18.204.196.117:/home/ec2-user](mailto:ec2-user@18.204.196.117:/home/ec2-user)

Copy from /home/ec2-user to local drive (current directory in the local drive).

sudo scp -i Mat-internship-ticket-key.pem ec2-user@54.175.5.201:/home/ec2-user/lynis\_audit\_2023-02-20\_16:45:06.log ./

Copy a directory from cli to local

scp -r user@<aws-instance-public-ip>:~/example /path/to/local/directory

copy from local to aws

scp -r mydir/ user@ip:/path/to/destination/

connecting to RDS instance:

copy keypair to an EC2 instance

sudo scp -r -i Mat-internship-ticket-key.pem ./Mat-internship-ticket-key.pem ec2-user@18.206.154.137:/home/ec2-user

ssh to the EC2 instance:

ssh -A -i "Mat-internship-ticket-key.pem" ec2-user@ec2-18-206-154-137.compute-1.amazonaws.com

check if 3306 is used in the EC2 instance

sudo lsof -i :3306

if used, try another port say 3307

ssh to DB instance:

sudo ssh -i "Mat-internship-ticket-key.pem" -L 3307:crm.cay0j1g5zceh.us-east-1.rds.amazonaws.com:3306 [ec2-user@18.206.154.137](mailto:ec2-user@18.206.154.137)

to grant previlages:

mysql -uroot -p

enter empty string – blank when prompt for password

MariaDB [(none)]> GRANT ALL PRIVILEGES ON \*.\* TO 'admin'@'crm.cay0j1g5zceh.us-east-1.rds.amazonaws.com' WITH GRANT OPTION;

You should receive:

**Query OK, 0 rows affected (0.00 sec)**

connect to MariaDB monitor (db user name = admin)

mysql -h 127.0.0.1 -P 3307 -u admin -p

enter DB password

MySQL prompt appears:

MySQL [(none)]>

Each command should end by “;”

Recall from history – list (say line number 85:

!85 :p

# Building Web Apps With Python Has Never Been Easier — Get Started With Pynecone

<https://medium.com/codingthesmartway-com-blog/building-web-apps-with-python-has-never-been-easier-get-started-with-pynecone-a9f60c1532c>

# Terraform Deploy a Two Tier Architecture in AWS

<https://levelup.gitconnected.com/terraform-two-tier-architecture-66fa07c8e325>

CDK example

<https://catalog.us-east-1.prod.workshops.aws/workshops/071bbc60-6c1f-47b6-8c66-e84f5dc96b3f/en-US>

Checkmk agent installation:

sudo yum install git

sudo git clone <https://github.com/tribe29/checkmk.git>

cd checkmk/agents

sudo wget <https://github.com/tribe29/checkmk/releases/download/1.6.0p23/check-mk-agent-1.6.0p23-1.noarch.rpm>

sudo yum install check-mk-agent-1.6.0p23-1.noarch.rpm

sudo systemctl start check-mk-agent

sudo systemctl enable check-mk-agent

Setting up a Checkmk

3. Launch an EC2 **Ubuntu 20.04** instance and give it the name of “Checkmk”. Create key pair if  
needed. Allow HTTP and under the Advanced details section > IAM Instance Profile select the  
instance profile you recently created in the SSM steps. All other defaults are OK. Click on Launch  
Instance Button.  
4. Navigate back to EC2 instances and select the Checkmk instance.  
5. Click on Connect  
6. Click on the Session Manager tab and then the Connect button.  
7. You will now be at the server terminal to begin Checkmk configurations

4. Sudo to root:  
sudo -i  
apt-get update  
5. To download Checkmk for Ubuntu 20.04 type in the command:  
wget https://download.checkmk.com/checkmk/2.1.0p18/check-mk-raw-2.1.0p18\_0.focal\_amd64.deb  
6. Install the Checkmk package:  
sudo apt install ./check-mk-raw-2.1.0p18\_0.focal\_amd64.deb  
7. Test installation with the command

omd version  
Sample output:  
OMD - Open Monitoring Distribution Version 2.1.0p18.cre

CDK installation in EC2 instance

sudo curl -sL https://rpm.nodesource.com/setup\_14.x | sudo bash -

sudo yum install -y nodejs

node -v

npm -v

sudo npm install -g aws-cdk

cdk –version

mkdir (new directory)

cd to the directory

cdk init --language=python

To answer Mathewos' question on the if block..... if \_\_name\_\_ == "\_\_main\_\_":

it is generally used if you are going to run the python file as a script as opposed to a module

[9:11](https://yellowtailtech-hub.slack.com/archives/C04J1QFUSTV/p1677636679160149)

a module is a file that can be imported by another python file. like how we import  module such as boto3, you are able to import a custom module that you create yourself. this is usually done when you are developing a much larger software program

[9:14](https://yellowtailtech-hub.slack.com/archives/C04J1QFUSTV/p1677636899478139)

so in our case, as we write code for automation scripts you will use the

if \_\_name\_\_ == "\_\_main\_\_": more frequently

[9:17](https://yellowtailtech-hub.slack.com/archives/C04J1QFUSTV/p1677637060739949)

in short the primary use case is control of having the file run as a main program/script OR imported by other modules

hi guys, hope everyone is enjoying their Wednesday! There was another question last night on how to run boto3 outside of the AWS account (ie: local terminal, not Cloud9, etc). So externally you would set up your config and credentials file like you normally would. these files would contain your profile names and secret key id and secret access keys.Now in your boto3 code you would reference these credentials so your script can establish a authenticated request to the AWS API. Few ways to handle that. First you can pass credentials directly as a parameterie:  
client = boto3.client('s3', aws\_access\_key\_id=ACCESS-KEY-HERE,  aws\_secret\_access\_key=SECRET-ACCESS-HERE, aws\_session\_token=SESSION-TOKEN-HERE)

another was to establish a session is to use the boto3 setup\_default\_session method. this requires the parameter of a profile\_name that you have set in your config/credentials file  
ie:  
boto3.setup\_default\_session(profile\_name="MY ACCOUNT PROFILE NAME")with that setting the default session you can now make your client connection to the AWS API and this will connect you to the specific account  
client = boto3.client('s3')

If aws-cli is not installed you can set your local Environment variables to point to your .aws/credentials or config file  
AWS\_SHARED\_CREDENTIALS\_FILE  
AWS\_CONFIG\_FILE

<https://www.hackerrank.com/onboarding?redirect=%2Fdashboard>

<https://res.cloudinary.com/acloud-guru/image/fetch/c_thumb,f_auto,q_auto/https://acg-wordpress-content-production.s3.us-west-2.amazonaws.com/app/uploads/2020/11/terraform-cheatsheet-from-ACG.pdf>

<https://acloudguru.com/blog/engineering/the-ultimate-terraform-cheatsheet>

TKT 14:

Top of Form

# Ticket 15. Collect custom metrics from your bastion host

Bottom of Form

Top of Form

Bottom of Form

# [ TASK ] You been assign to create an Ansible playbook to install the CloudWatch agent on all bastion hosts. The playbook should also put custom metric such as memory and disk space from the bastion host.

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-scripts-intro.html>

1. Installed the required package in i-08c615c903c34b1d6 instance.

sudo yum install -y perl-Switch perl-DateTime perl-Sys-Syslog perl-LWP-Protocol-https perl-Digest-SHA.x86\_64

2. Downloaded/installed/configured the monitoring script:

curl <https://aws-cloudwatch.s3.amazonaws.com/downloads/CloudWatchMonitoringScripts-1.2.2.zip> -O

Unzipped:

unzip CloudWatchMonitoringScripts-1.2.2.zip && \  
rm CloudWatchMonitoringScripts-1.2.2.zip && \  
cd aws-scripts-mon

3. Cloudwatch and EC2 instance permissions added in the role - “Role-Ticket-MAT-49”

4. Performed a simple test without positing data:

./mon-put-instance-data.pl --mem-util --verify --verbose

5. Collected all available memory metrics and sent them to CloudWatch, counting cache and buffer memory as used

./mon-put-instance-data.pl --mem-used-incl-cache-buff --mem-util --mem-used --mem-avail

6. Collected aggregated metrics for an Auto Scaling group and sent them to Amazon CloudWatch without reporting individual instance metrics.

./mon-put-instance-data.pl --mem-util --mem-used --mem-avail --auto-scaling=only

7. Collected aggregated metrics for instance type, AMI ID and region, and sent them to Amazon CloudWatch without reporting individual instance metrics

./mon-put-instance-data.pl --mem-util --mem-used --mem-avail --aggregated=only

7. crontab -e (added with the following to report memory and disk utilization every 5 minutes)

\*/5 \* \* \* \* ~/aws-scripts-mon/mon-put-instance-data.pl --mem-used-incl-cache-buff --mem-util --disk-space-util --disk-path=/ --from-cron

8. Checked utilization statistics in the last 12 hrs.

./mon-get-instance-stats.pl --recent-hours=12

Ansible playbook yaml script:

---

- name: Install CloudWatch agent and send custom metrics on bastion hosts

  hosts: local

  gather\_facts: false

  tasks:

- name: Gather a list of instances with the tag "EC2-TKT-49-54-Bastion Instance"

    ec2\_instance\_info:

      filters:

        "tag:Name": "EC2-TKT-49-54-Bastion Instance"

    register: bastion\_instances

- name: Install dependencies

    package:

      name: awslogs

      state: present

    delegate\_to: "{{ item.private\_ip }}"

    loop: "{{ bastion\_instances.instances }}"

- name: Download the CloudWatch agent configuration file

    command: aws s3 cp s3://my-cloudwatch-bucket/cloudwatch-agent-config.json /tmp/cloudwatch-agent-config.json

    delegate\_to: "{{ item.private\_ip }}"

    loop: "{{ bastion\_instances.instances }}"

- name: Install the CloudWatch agent

    command: /usr/local/bin/aws cloudwatch configure-agent --input-path /tmp/cloudwatch-agent-config.json

    delegate\_to: "{{ item.private\_ip }}"

    loop: "{{ bastion\_instances.instances }}"

- name: Start the CloudWatch agent

    service:

      name: awslogs

      state: started

      enabled: yes

    delegate\_to: "{{ item.private\_ip }}"

    loop: "{{ bastion\_instances.instances }}"

- name: Send custom metrics - Memory Usage

    shell: echo "{{ item.private\_ip }} memory\_usage `free -m | grep Mem | awk '{print $3/$2 \* 100.0}'`" | /usr/local/bin/aws cloudwatch put-metric-data --metric-name MemoryUsage --unit Percent --value `free -m | grep Mem | awk '{print $3/$2 \* 100.0}'` --dimensions InstanceId={{ item.instance\_id }}

    delegate\_to: "{{ item.private\_ip }}"

    loop: "{{ bastion\_instances.instances }}"

- name: Send custom metrics - Disk Space Utilization

    shell: echo "{{ item.private\_ip }} disk\_utilization `df -h / | grep / | awk '{print $5}' | sed 's/%//g'`" | /usr/local/bin/aws cloudwatch put-metric-data --metric-name DiskSpaceUtilization --unit Percent --value `df -h / | grep / | awk '{print $5}' | sed 's/%//g'` --dimensions InstanceId={{ item.instance\_id }}

    delegate\_to: "{{ item.private\_ip }}"

    loop: "{{ bastion\_instances.instances }}"

Python assignment lambda function

lambda.py

#!/usr/bin/python3

import boto3

import json

iam = boto3.client('iam')

role\_policy = {

"Version": "2012-10-17",

"Statement": [

{

"Sid": "",

"Effect": "Allow",

"Principal": {

"Service": "lambda.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

]

}

try:

response = iam.get\_role(RoleName='LambdaBasicExecution')

print('Role already exists')

except iam.exceptions.NoSuchEntityException:

response = iam.create\_role(

RoleName='LambdaBasicExecution',

AssumeRolePolicyDocument=json.dumps(role\_policy),

)

print('Role created successfully')

print(response)

the only way I found to clear the error -  
***"botocore.exceptions.ClientError: An error occurred (InvalidClientTokenId) when calling the CreateRole operation: The security token included in the request is invalid"*** is to use export in the CLI as below and entering the access key and secret access key.  
mathewos-viti:~/environment $ export AWS\_ACCESS\_KEY\_ID=xxxxx  
mathewos-viti:~/environment $ export AWS\_SECRET\_ACCESS\_KEY=yyyy

handler.py

#!/usr/bin/python3

import boto3

import json

def lambda\_handler(event, context):

response = {

'statusCode': 200,

'body': json.dumps('Hello from Lambda!')

}

return response

# Call the lambda\_handler function

result = lambda\_handler(None, None)

# Print the JSON response

print(result['body'])

to zip the CLI command

zip lambda\_function.zip handler.py

lambda2.py

#!/usr/bin/python3

import boto3

iam\_client = boto3.client('iam')

lambda\_client = boto3.client('lambda')

with open('lambda.zip', 'rb') as f:

zipped\_code = f.read()

role = iam\_client.get\_role(RoleName='LambdaBasicExecution')

response = lambda\_client.create\_function(

FunctionName='helloWorldLambda',

Runtime='python3.9',

Role=role['Role']['Arn'],

Handler='handler.lambda\_handler',

Code=dict(ZipFile=zipped\_code),

Timeout=300, # Maximum allowable timeout

)

print(response)

lambda3.py

#!/usr/bin/python3

# Define the name of the output ZIP file

my\_lambda\_functionzip = 'lambda\_function.zip'

# Specify the name of the directory containing the Lambda function code

my\_lambda\_function = '/home/ec2-user/environment/lambda\_function' #all files are under this directory

import shutil

# Create a ZIP archive of the Lambda function code and the handler module

shutil.make\_archive(my\_lambda\_functionzip, 'zip', my\_lambda\_function, 'handler.py')

lambda4.py

#!/usr/bin/python3

import boto3, json

lambda\_client = boto3.client('lambda')

test\_event = dict()

response = lambda\_client.invoke(

FunctionName='helloWorldLambda',

Payload=json.dumps(test\_event),

)

print(response['Payload'])

print(response['Payload'].read().decode("utf-8"))

lambda5.py

#!/usr/bin/python3

import boto3, json

import boto3

lambda\_client = boto3.client('lambda')

response = lambda\_client.get\_function(

FunctionName='helloWorldLambda'

)

print(response)

lambda6.py

#!/usr/bin/python3

import boto3, json

import boto3

lambda\_client = boto3.client('lambda')

response = lambda\_client.add\_permission(

StatementId='S3InvokeHelloWorldLambda',

FunctionName='helloWorldLambda',

Action='lambda:InvokeLambda',

Principal='s3.amazonaws.com',

SourceArn='arn:aws:s3:::test-ap-s3-bucket/\*',

)

print(response)

lambda7.py

#!/usr/bin/python3

import boto3, json

import boto3

lambda\_client = boto3.client('lambda')

response = lambda\_client.delete\_function(

FunctionName='helloWorldLambda'

)

print(response)

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-scripts-intro.html>

Ansible playbook abcdefg.yaml script:

---

- name: Install CloudWatch agent and send custom metrics on bastion

hosts hosts: local

gather\_facts: false

tasks:

- name: Gather a list of instances with the tag "EC2-TKT-49-54-Bastion Instance"

ec2\_instance\_info:

filters:

"tag:Name": "EC2-TKT-49-54-Bastion Instance"

register: bastion\_instances

- name: Install dependencies

package:

name: awslogs

state: present

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Download the CloudWatch agent configuration file

command: aws s3 cp s3://my-cloudwatch-bucket/cloudwatch-agent-config.json /tmp/cloudwatch-agent-config.json

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Install the CloudWatch agent

command: /usr/local/bin/aws cloudwatch configure-agent --input-path /tmp/cloudwatch-agent-config.json

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Start the CloudWatch agent

service:

name: awslogs

state: started

enabled: yes

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Send custom metrics - Memory Usage

shell: echo "{{ item.private\_ip }} memory\_usage `free -m | grep Mem | awk '{print $3/$2 \* 100.0}'`" | /usr/local/bin/aws cloudwatch put-metric-data --metric-name MemoryUsage --unit Percent --value `free -m | grep Mem | awk '{print $3/$2 \* 100.0}'` --dimensions InstanceId={{ item.instance\_id }}

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

- name: Send custom metrics - Disk Space Utilization

shell: echo "{{ item.private\_ip }} disk\_utilization `df -h / | grep / | awk '{print $5}' | sed 's/%//g'`" | /usr/local/bin/aws cloudwatch put-metric-data --metric-name DiskSpaceUtilization --unit Percent --value `df -h / | grep / | awk '{print $5}' | sed 's/%//g'` --dimensions InstanceId={{ item.instance\_id }}

delegate\_to: "{{ item.private\_ip }}"

loop: "{{ bastion\_instances.instances }}"

This Ansible playbook performs the following tasks:

1. Gathers a list of EC2 instances that have a specific tag.
2. Installs the CloudWatch agent dependencies on each instance.
3. Downloads the CloudWatch agent configuration file from an S3 bucket on each instance.
4. Installs the CloudWatch agent on each instance using the configuration file.
5. Starts the CloudWatch agent on each instance.
6. Sends custom metrics to CloudWatch for each instance, specifically the memory usage and disk space utilization.

Here's a brief explanation of the different parts of the playbook:

* The **name** keyword is used to give a name to the playbook, which is purely descriptive.
* The **hosts** keyword specifies the hosts that the playbook should run on. In this case, the hosts are defined as **local**, which means that Ansible will use the local connection plugin to connect to the hosts.
* The **gather\_facts** keyword is set to false, which means that Ansible will not gather facts about the hosts before running the playbook. This can be useful if you don't need to access the facts in your playbook.
* The **tasks** keyword contains a list of tasks to be executed. Each task has a **name** keyword that gives it a descriptive name, and a set of actions to perform.
* The **ec2\_instance\_info** module is used to gather information about EC2 instances that have a specific tag. The **register** keyword is used to store the results of the module in a variable called **bastion\_instances**.
* The **package** module is used to install the CloudWatch agent dependencies on each instance. The **delegate\_to** keyword is used to run the task on the remote host.
* The **command** module is used to download the CloudWatch agent configuration file from an S3 bucket on each instance. The **delegate\_to** keyword is used to run the task on the remote host.
* The **command** module is also used to install the CloudWatch agent on each instance using the configuration file. The **delegate\_to** keyword is used to run the task on the remote host.
* The **service** module is used to start the CloudWatch agent on each instance. The **delegate\_to** keyword is used to run the task on the remote host.
* The **shell** module is used to send custom metrics to CloudWatch for each instance, specifically the memory usage and disk space utilization. The **delegate\_to** keyword is used to run the task on the remote host. The **loop** keyword is used to iterate over the list of instances stored in the **bastion\_instances** variable, and the **item** variable is used to access the current instance in the loop.

Overall, this playbook automates the process of installing and configuring the CloudWatch agent on a group of EC2 instances, and sends custom metrics to CloudWatch for each instance.

To run this Ansible playbook, you can use the following command:

ansible-playbook <path\_to\_playbook\_file>

For example, if the playbook is saved in a file named **cloudwatch-agent.yml** in the current directory, you can run it using the command:

ansible-playbook cloudwatch-agent.yml

Make sure you have the necessary permissions to run the playbook and that your inventory file is properly configured.

735 repositories:

<https://github.com/awslabs/>