AWS Cloud Computing - Week 2 (CLI, ASG, AMI)

AWS CloudShell

- Browser based (Linux-based) shell in AWS console
- AWS tooling pre-installed (AWS CLIv2, ECS CLI, SAM CLI, runtimes and SDK for Python and Node.JS, shell tooling)
- 1 GB persistent storage in home directory up to 120 days
- Activity must be interactive (up to 12 hours) as lack of keyboard input will stop shell after 20 - 30 minutes (background jobs do not count)
- No charge
- Upload and download files
- No access to VPC resources
- No internet ingress, only egress
- Access control via IAM

Using `aws-shell`

- `aws-shell` does not work with AWS CLIv2.
- Project is abandoned and most functionality is integrated into AWS CLIv2.
- To get similar functionality from AWS CLIv2 as `aws-shell` do:

```
# Enable in AWS CLI configuration `~/.aws/config`
aws configure set cli_auto_prompt on
```

or

```
# Enable in current shell (can put in shell profile to set for every
login)
export AWS_CLI_AUTO_PROMPT=on
```

or

```
# Enable in current AWS CLI invocation
aws --cli-auto-prompt
```

AWS CLI

By itself not the best IaC and not considered IaC

- imperative (this is what I want to create)
- no language, just CLI with parameters
- no parameters (need to write wrapper script to inject values)
- no separating command from "code"

AWS CloudFormation, AWS CDK, Hashicorp Terraform, and Pulmni are examples of IaC.

- declarative (this is how I want to create)

Tutorial/Lab - Creating a VPC and EC2

Create VPC and full network

```
# Step 1: Create VPC
aws ec2 create-vpc \
--cidr-block 10.50.0.0/16
VPC_ID=$(aws ec2 describe-vpcs --filters Name=cidr,Values=["10.50.0.0/16"]
--query 'Vpcs[*].VpcId' --output=text)
# Add tags to VPC
aws ec2 create-tags \
--resources ${VPC_ID} \
--tags Key=Name,Value=demo-cli-vpc

# Setp 2a: Create (public) subnets
aws ec2 create-subnet \
--vpc-id ${VPC_ID} \
--cidr-block 10.50.1.0/24
PUB_SUBNET_ID=$(aws ec2 describe-subnets --query 'Subnets[?
CidrBlock==`10.50.1.0/24`].SubnetId' --output=text)
# Tag (public) subnet
aws ec2 create-tags \
--resources ${PUB_SUBNET_ID} --tags Key=Name,Value=demo-cli-pub-us-east-1

# Step 2b: Create (private) subnet
aws ec2 create-subnet \
--vpc-id ${VPC_ID} \
--cidr-block 10.50.2.0/24
PRV_SUBNET_ID=$(aws ec2 describe-subnets --query 'Subnets[?
CidrBlock==`10.50.2.0/24`].SubnetId' --output=text)
# Tag (private) subnet
aws ec2 create-tags \
--resources ${PRV_SUBNET_ID} --tags Key=Name,Value=demo-cli-prv-us-east-1

# Step 3: Create IGW
aws ec2 create-internet-gateway \
--tag-specifications ResourceType=internet-
gateway,Tags='[{Key="Name",Value="demo-cli-igw"}]'
IGW_ID=$(aws ec2 describe-internet-gateways --query 'InternetGateways[?
```

```
(Attachments==`[]` && Tags[?Value==`demo-cli-igw`])].InternetGatewayId' --
output text)
# Attach IGW to VPC
aws ec2 attach-internet-gateway \
--internet-gateway-id ${IGW_ID} \
--vpc-id ${VPC_ID}

# Step 4: Create Elastic IP for NAT Gateway
aws ec2 allocate-address \
--domain vpc \
--tag-specifications ResourceType=elastic-
ip,Tags='[{Key="Name",Value="demo-cli-eip"}]'
EIP_ALLOC_ID=$(aws ec2 describe-addresses --query 'Addresses[?Tags[?
Value==`demo-cli-eip`]].AllocationId' --output text)
# Create NAT Gateway
aws ec2 create-nat-gateway \
--subnet-id ${PUB_SUBNET_ID} \
--allocation-id ${EIP_ALLOC_ID} \
--tag-specifications
ResourceType=natgateway,Tags='[{Key="Name",Value="demo-cli-natgw-us-east-
1"}]'
NATGW_ID=$(aws ec2 describe-nat-gateways --query "NatGateways[?
VpcId==\`${VPC_ID}\`].NatGatewayId" --output=text)

# Step 5a: Create (public) Route Table
aws ec2 create-route-table \
--vpc-id ${VPC_ID} \
--tag-specifications ResourceType=route-
table,Tags='[{Key="Name",Value="demo-cli-pub-rt"}]'
PUB_RT_ID=$(aws ec2 describe-route-tables --query 'RouteTables[?Tags[?
Value==`demo-cli-pub-rt`]].RouteTableId' --output=text)

# Step 5b: Create (private) Route Table
aws ec2 create-route-table \
--vpc-id ${VPC_ID} \
--tag-specifications ResourceType=route-
table,Tags='[{Key="Name",Value="demo-cli-prv-rt"}]'
PRV_RT_ID=$(aws ec2 describe-route-tables --query 'RouteTables[?Tags[?
Value==`demo-cli-prv-rt`]].RouteTableId' --output=text)
```

```
# Step 6: Create routes
aws ec2 create-route \
--route-table-id ${PUB_RT_ID} \
--destination-cidr-block 0.0.0.0/0 \
--gateway-id ${IGW_ID}
aws ec2 create-route \
--route-table-id ${PRV_RT_ID} \
--destination-cidr-block 0.0.0.0/0 \
--gateway-id ${NATGW_ID}

# Step 7: Associate RT to subnet
# Associate (public) subnet and RT
aws ec2 associate-route-table \
--route-table-id ${PUB_RT_ID} \
--subnet-id ${PUB_SUBNET_ID}
# Associate (private) subnet and RT
aws ec2 associate-route-table \
--route-table-id ${PRV_RT_ID} \
--subnet-id ${PRV_SUBNET_ID}

# Step 8: Create SG for VPC
aws ec2 create-security-group \
--description "Allow lab access" \
--group-name demo-cli-sg \
--vpc-id ${VPC_ID} \
--tag-specifications ResourceType=security-
group,Tags='[{Key="Name",Value="demo-cli-sg"}]' # Obtain SG ID to be used
in subsequent commands
SG_ID=$(aws ec2 describe-security-groups --query 'SecurityGroups[?Tags[?
Value==`demo-cli-sg`]].GroupId' --output text)

aws ec2 authorize-security-group-ingress \
--group-id ${SG_ID} \
--protocol tcp \
--port 22 \
--cidr $(curl https://checkip.amazonaws.com)/32 \
--tag-specifications ResourceType=security-group-
rule,Tags='[{Key="Name",Value="demo-cli-in-tcp22-sgr"}]'
aws ec2 authorize-security-group-ingress \
--group-id ${SG_ID} \
```

```
--protocol tcp \
--port 80 \
--cidr $(curl https://checkip.amazonaws.com)/32 \
--tag-specifications ResourceType=security-group-
rule,Tags='[{Key="Name",Value="demo-cli-in-tcp80-sgr"}]'
```

Create/Run EC2

Step 1: Create SSH Key Pair (optional, prefer to use Session Manager)

```
aws ec2 create-key-pair \
--key-name demo-cli-us-east-1-kp \
--query 'KeyMaterial' \
--tag-specifications ResourceType=key-pair,Tags='[{Key="Name",Value="demo-
cli-us-east-1-kp"}]' \
--output text > ~/.ssh/demo-cli-us-east-1-kp.pem
chmod 400 ~/.ssh/demo-cli-us-east-1-kp.pem
```

Step 2: Launch EC2

Option 1: CLI

```
aws ec2 run-instances \
--image-id ami-033b95fb8079dc481 \
--count 1 \
--instance-type t2.micro \
--key-name demo-cli-us-east-1-kp \
--subnet-id ${PUB_SUBNET_ID} \
--security-group-ids ${SG_ID} \
--associate-public-ip-address \
--key-name demo-cli-us-east-1-kp \
--tag-specifications ResourceType=instance,Tags='[{Key="Name",Value="demo-
cli-ec2"}]'
INSTANCE_ID=$(aws ec2 describe-instances --query
'Reservations[*].Instances[?Tags[?Value==`demo-cli-ec2`]].InstanceId' --
output=text)
```

Option 2: CFN

```
aws cloudformation deploy \
--stack-name demo-cli-cfn-ec2 \
--template-file template.cfn.yaml \
```

```
--parameter-overrides KeyName=demo-cli-us-east-1-kp InstanceType=t2.micro
\
--capabilities CAPABILITY_IAM
```

Obtain EC2 IP and SSH

```
INSTANCE_IP=$(aws ec2 describe-instances \
--instance-ids {$INSTANCE_ID} \
--query 'Reservations[*].Instances[*].[PublicIpAddress]' \
--output text)
ssh -i ~/.ssh/lab-us-east-1-kp.pem ec2-user@${INSTANCE_IP}
```

Setup app

```
sudo su
APP_NAME=LiftShift-Application
yum update -y && yum -y install python3-pip zip
wget https://d6opu47qoi4ee.cloudfront.net/loadbalancer/simuapp-v1.zip
unzip simuapp-v1.zip
sed -i "s=MOD_APPLICATION_NAME=$APP_NAME=g" templates/index.html
pip3 install -r requirements.txt
nohup python3 simu_app.py >> application.log 2>&1 &
```

Test ASG Behavior

- Use a CFN template which creates ASG workload (easy button).
- Put some CPU load on the server by executing on the server: `stress --cpu 4 --timeout 300`

Cleanup

```
# Step 1: Terminate instance
aws ec2 terminate-instances \
--instance-ids ${INSTANCE_ID}

# For CFN: aws cloudformation delete-stack --stack-name demo-cli-cfn-ec2

# Step 2: Delete SG
aws ec2 delete-security-group \
--group-id ${SG_ID}
```

```
# Step 3: Delete KP
aws ec2 delete-key-pair \
--key-name demo-cli-us-east-1-kp

# Step 4: Delete NATGW
aws ec2 delete-nat-gateway --nat-gateway-id $NATGW_ID

# Step 5: Delete EIP
aws ec2 release-address --allocation-id ${EIP_ALLOC_ID}

# Step 6: Delete subnets
aws ec2 delete-subnet --subnet-id ${PRV_SUBNET_ID}
aws ec2 delete-subnet --subnet-id ${PUB_SUBNET_ID}

# Step 7: Delete RTs
aws ec2 delete-route-table --route-table-id ${PUB_RT_ID}
aws ec2 delete-route-table --route-table-id ${PRV_RT_ID}

# Step 8: Detach IGW
aws ec2 detach-internet-gateway --internet-gateway-id ${IGW_ID}

# Step 9: Delete IGW
aws ec2 delete-internet-gateway --internet-gateway-id ${IGW_ID}

# Step 10: Delete VPC
aws ec2 delete-vpc --vpc-id ${VPC_ID}
```

AMI

Amazon Machine Image, is an image of an OS that is used to quickly launch instances without configuration (unless desired during bootstrap).

- You can buy (AWS Marketplace), share (within the same account or across accounts, across regions too), and sell (AWS Marketplace) AMIs
- Can have different root volumes types: EBS (talk later) which is attached from storage network, instance store which is tied to server (fastest, but less flexible than EBS).
- Create AMIs using AWS tools, HashiCorp Packer, or other tooling

Auto Scaling Group

A few times of scaling:

- Manual Scaling: you control everything, how to scale, when to scale, how much, adding/removing instances, etc. Not recommended.
- Dynamic Scaling: the traditional scaling type where it supports:
  - Target tracking: uses CloudWatch metric and target value to scale (think thermostat)
  - Step scaling: uses step adjustments (increment/decrement set number of instances) based on size of threshold breach.
  - Simple scaling: uses single scaling adjustment (such as CPU >= 75% utilization) with cooldown period between scaling events (to avoid fluctuations). This is the original
- Scheduled Scaling: set recurring schedule on which to scale
- Predictive Scaling: Similar to above but less hands-on using machine learning. We will go through next

Some best practices

- Enable detailed monitoring (additional cost) to get CloudWatch metric data for EC2 instances at a one-minute frequency because that ensures a faster response to load changes
- Enable Auto Scaling group metrics, otherwise, actual capacity data is not shown in the capacity forecast graphs that are available on completion of the Create Scaling Plan wizard.
- Use appropriate instance types. For example, using t2/3 which offer baseline CPU and burst to higher levels based on workload can exhaust CPU credits resulting in decreased performance at that point. Can mitigate somewhat by specifying as “unlimited” to use surplus credits.
- Use lifecycle hooks to add functionality during instance events. Such as download logs during instance termination event for forensics (termination event is suspended), or execute domain join script during instance launch event. Wait period is finite, see docs for details.

Predictive Scaling

- Good for pattern based workloads. Cyclical or repeating in nature. Uses CloudWatch metric data.
- Uses last 14 days of data to predict next 48 hours
- Predictive Scaling is dependent on quality of forecast model for cyclical nature of workload. Evaluate Predictive Scaling by running in forecast only mode, then in forecast and scale model if happy with forecast quality.
- Use multiple policies in forecast mode to get best option to scale
- Can use predefined or custom metrics (metrics you publish to CloudWatch)
- Use in combination with Dynamic Scaling
- Use in combination with Scheduled Actions (Scheduled Scaling) to account for known events outside of pattern (such as upcoming marketing event)

Warm Pools

Instances take time to bootstrap and come online, so to avoid this wait period and its affect on latency and workload processing, you specify to keep a set of instances pre-initialized for quicker availability during scale out events (i.e. they are added to the desired count of the ASG). The default size is max - desired.

Actual instance state in warm pool is either "stopped", "running", or "hibernated".

- warm pool can also save cost by avoiding over-provisioning of ASG, but be aware:
  - "stopped" state is recommended to save cost but is slowest to be ready, and EBS volumes are still charged
  - "hibernated" is next quickest to ready state and similar to "stopped" saves money, but must be supported by the OS and hypervisor
  - "running" state is quickest, but is most costly

Useful Links

- Sample CloudFormation templates: https://github.com/awslabs/aws-cloudformation-templates
- SG Connection Tracking: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-rules.html
- Systems Manager Session Manager: https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html

#great-learning #mentoring