AWS

suredatla@gmail.com / standard password -> root user

Alias: suredatla

12 digit account Id: 478300270166

suresh / S....h@... -> new admin user

alias: <https://suredatla.signin.aws.amazon.com/console>

----------------------------------------------------------------------------------------------------

Regions:

* Cluster of Data Centers. Has name and code. Eg: Asia Pacific (Mumbai) & ap-south-1

Availability Zones/Zones (AZs):

* Each region can have many Availability Zones. At a minimum, there would be 2 AZs per region.
* E.g.: ap-south-1 region has ap-south-1a, ap-south-1b and ap-south-1c AZs.
* Each AZ is one or more distinct data-centers
* AZs are geographically isolated from each other, by a meaningful distance, but within 100km.
* AZs are connected with each other with high bandwidth and ultra-low latency networking.

IAM (Identity and Access Management):

* Has Users, Groups, Roles
* Users would be categorized as Groups and Roles is assigned to AWS resources.
* Policies (which are JSON documents) determines what Users, Groups and Roles can/cannot do.
* Large enterprises integrate their own repository of users, such as Active Directory, with IAM using AWS SSO or IAM. AWS SSO is a great choice to help you define federated access permissions for your users based on their group memberships in a single centralized directory. If you use multiple directories, or want to manage the permissions based on user attributes, consider AWS IAM as your design alternative. AWS SSO works with an Identity Provider (IdP) of your choice, such as Okta Universal Directory or Azure Active Directory (AD) via the [Security Assertion Markup Language 2.0](https://aws.amazon.com/iam/details/saml/) (SAML 2.0) protocol.
* Guidelines: 1 IAM User per physical person; 1 IAM role per Application; IAM credentials should never be shared and should never be in code (leads to misuse like bitcoin); never use the root account except for initial setup.

EC2 (Elastic Compute Cloud):

* Includes the capabilities of
  + Launching virtual machines on the cloud (EC2)
  + Storing data on virtual cloud (EBS)
  + Distributing load across machines (ELB)
  + Scaling services using an auto scaling group (ASG)
* This was used before Serverless stepped-in
* Creating an Instance:
  + Choose an Amazon Machine Image (AMI) – template that contains the SW configuration (OS, Application Server, and Apps) required to launch our instance.
  + Choose an instance type (t2.nano, t2.micro, t2.small, t2.medium, t2.large, t2.xlarge, t2.2xlarge,…) determines the power of instance in terms of vCPUs & Memory (GiB)
  + Configure Instance Details:
    - A default Amazon Virtual Private Cloud (VPC) would be available for use, we can instead create our own VPC to select our own IP address range, to create subnets, to configure route tables and to configure network gateways. Can choose the available subnets or create a new one, each subnet resides in one AZ.
    - Can add storage, OS would be installed in root volume storage that would be configured by default.
    - Can add tags (key-value pairs) e.g. name tag can be defined.
    - Can configure security-group. A security group is a set of firewall rules that control the traffic for your instance. You can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one. In Linux, we should be able to do SSH into it. If Source is set to 0.0.0.0/0 it means we allow all IP addresses to access our instance.
    - Create new/choose existing Key Pair: A key pair consists of a public key that AWS stores, and a private key file that we store. Together, they allow us to connect to our instances securely. For windows AMIs, the private key file is required to obtain the password used to log into our instance. For Linux AMIs, the private key file allows us to securely SSH into our instance. Create a new Key-Pair and download that Key-Pair (\*.pem file), we will not be able to download the file again after it’s created.

Connecting to EC2 Instance: Through SSH/Putty/EC2 Instance Connect

* ssh **-i** my-ec2-key-pair.pem **ec2-user**@13.126.11.27 (ssh -i <key pair file> ec2-user@<public-ip>), i stands for identity file.
* In Puttygen, load the key-pair file and save it as .ppk file. Then use putty application. In host-name, provide ec2-user@<public-ip> (e.g. ec2-user@13.126.11.27). Link the .ppk file this way (Connection->SSH->Auth->browse and choose private key file), Go to Session and click on save, before clicking ‘Open’.

Security Groups:

* SSH is on TCP protocol, on port 22
* Can control inbound and outbound traffic
* Has many-to-many relationship with EC2 instances
* Is tied to region/VPC combination
* Best Practice: To maintain one separate security group just for SSH access
* When the connection to EC2 times-out, it would be due to issue with security group.
* Connection-refused error would be related to application/launch error
* All inbound traffic is blocked by default; All outbound traffic is authorized by default
* Referencing Security Groups from other Security Groups: In Inbound, we can say we are authorizing a specific security-group. If we launch another EC2 instance which has this specific security-group attached to it, we are basically allowing the other EC2 instance to connect to our first EC2 instance on the port we are allowing, regardless of our IP. **How to do this, practical required**
* Range of IPV4 addresses in the form of 10.0.0.0/16, are called CIDR blocks (Classless Inter-Domain Routing)

IPs:

* Networking has two sorts of IPs (IPV4, IPV6). IPV4 is 4 numbers separated by 3 dots (e.g. 1.140.10.240). IPV4 allows 3.7 billion addresses in public space, [0..255]. [0..255]. [0..255]. [0..255]. IPV6 is long hexadecimal numbers (e.g. 1900:4545:3:200:f8ff:fe21:67cf). AWS supports IPV6 as well. IPV6 is common for Internet of Things. Public IP can be geo-located easily.
* Private Network has private IP range. Within private network, every computer can talk to one another, and can also have an internet gateway with a public IP to talk to other servers. Two different private networks can have the same private IP. **Machines which are on private network would connect to internet through a NAT device and internet gateway that will act as a proxy**.
* If we need to have a fixed public IP for our EC2 instance, we need an elastic IP, which we own it as long as we don’t delete it. Can be attached to one instance at a time. We can have only 5 elastic IPs in our account, can request for more. Using elastic IP, reflects poor architectural decision. Instead, use a random public IP and register a DNS name to it.
* By default, EC2 machine comes with a private IP and a public IP.
* When we restart an EC2 instance, the public IP changes, but the private IP does not change
* Can create a new Elastic-IP, by choosing “Allocate new address” in “Network & Security -> Elastic-IP”. Can associate elastic-IP with EC2 instance, by choosing “Associate”, and can disassociate by choosing “Disassociate”. Though elastic-IPs are free, if there is any elastic-IP which is not associated with any running instance, charges would be applicable.

Installing Apache Web Server on EC2 instance:

* sudo su (SuperUserDo SwitchUser), allows us to run commands with privileges of another user, root user by default. Run ‘sudo su’ to install web server. From ec2-user, we would switch to root user (can verify by running whoami)
* Run ‘yum update -y’ , which forces the machine to update itself. ‘-y’ is to allow the updates to go through without prompting for allow/disallow. Yum is the primary tool for getting, installing, deleting, querying, and managing RHEL SW packages from Red Hat repositories.
* Installing httpd -> yum install -y httpd.x86\_64
* To start httpd service -> systemctl start httpd.service (systemctl is a controlling interface and inspection tool for the widely adopted init system and service manager system)
* To enable httpd service across reboots -> systemctl enable httpd.service
* Doing ‘curl localhost:80’ would display the test web page (curl is to load the content of web url; 80 is the port assigned for HTTP)
* Add inbound rule in the security-group of EC2 instance, to allow HTTP requests on port 80, else the web requests will time out without any response. (0.0.0.0/0 is for allowing all IPV4 traffic, ::/0 is to allow all IPV6 traffic)
* Our content can be added to “/var/www/html/” as mentioned in the test-page of Apache server -> e.g. echo “Hello World from $(hostname -f)” > /var/www/html/index.html

($(hostname -f) returns the internal DNS, with internal IP address)

EC2 user data:

* Using EC2 user data script, we can bootstrap EC2 instances, i.e. launch commands when a machine starts.
* While creating EC2 instance, during ‘Step3 – Configure Instance Details’, under ‘Advanced Details’, we have ‘User Data’

Sample: #!/bin/bash -> Mandatory (gets base 64 encoded)

yum update -y -> <list all the commands that we wish to>

yum install -y httpd.x86\_64

systemctl start httpd.service

systemctl enable httpd.service

EC2 Launch Types:

* On-Demand Instances: short workload, predictable pricing. Pay for what we use, billing per second, after first minute. Has the highest cost, but no upfront payment. For short-term needs and for un-interrupted and elastic workloads.
* Reserved Instances: If our need is to use the instance for at-least a year or so. Can be ‘Reserved’, ‘Convertible Reserved’ or ‘Scheduled Reserved’. Convertible instances would have flexible instances, today we can have m4.xlarge today but could be c5.large tomorrow. Example of scheduled reserved instance use-case, ‘every Thursday between 3 and 6pm’. Can get upto 75% discount when compared to On-Demand. Can reserve for 1 or 3 years. With ‘Convertible Reserved’, can get discount upto 54%.
* Spot Instances: short workload, for cheap, can lose instances, so less reliable.
* Can get upto 90% discount when compared to on-demand. UseCases: Batch-job that we can retry, data-analysis, etc. Anything that’s possible to retry can be a good use-case for spot instance.
* If current spot price > our max price, we can choose to stop or terminate our instance within 2 min grace period. If we stop, then the day when the current spot price < our max price, we can start the instance and continue from where we left it off.
* “Spot Block” strategy: To block spot instance during a specified time frame (1 to 6hrs) without interruptions. With ‘Spot Block’, in rare instances the instance will be reclaimed. Pricing of “Spot Block” differs from “Spot Instance” pricing.
* We pay the spot price that’s in effect at the beginning of each instance-hour for our running instance, billed to the nearest second. We never pay more than the maximum price we specify.
* Right order of terminating the spot-instances. Cancel the open spot-request (if it exists) corresponding to the active spot-instance and then cancel the spot-instance, so that the spot-request does not start a new spot-instance.
* Spot Fleets – Set of spot instances & optional-on-demand-instances. Should define launch pools (characterized by instance-type, OS, AZ), and fleet would choose from these launch pools. Spot fleet will try to meet the target capacity with price constraints. Stops launching instances when capacity/max-budget is reached. Can go for one of these strategies: lowestPrice, diversified, capacityOptimized.
* For Spot Fleet request needs, can use “Spot Request” link. For one spot-instance need, can choose “Request Spot Instances” checkbox during “Step3: Configure Instance Details” during Launch Instance.
* Dedicated Instances: No other customer would share the HW. May share HW with other instances in same account. Has no control over instance placement.
* Dedicated hosts: Book an entire physical server, control instance placement. Visibility into underlying sockets/physical cores of HW, which is great for licensing purposes. We get it for 3yr period reservation. This would be required when we have SW that has complicated licensing model, and for companies that have strong regulatory/compliance needs. During “Launch Instance”, in “Step3: Configure Instance Details”, can select the appropriate option in Tenancy dropd-down: Shared/Dedicated/Dedicated host.
* Best practice: Use reserved instances for baseline capacity. For anything that’s unpredictable, for peaks, can use mix of spot-instances and on-demand instances, basis the failure resiliency.

EC2 Instance Types:

* R: Apps that need lot of RAM/in-memory caches
* C: Apps that need good CPU – Compute/Database/BigData
* M: Middle of R & C – General/Web App
* I: Apps that need good local I/O – Instance Storage/Disc Operations/Database
* G: Apps that need GPU – Video Rendering/Machine Learning
* T2/T3: Burstable instances (up to a capacity) – We get a good performance for a burst/short-while, but if we over abuse that burst, we loose the burst/capacity. When the machine needs to process something unexpected (e.g. spike in load), it can burst and CPU can be very good. When the machine bursts, it utilizes “burst credits”. If all credits are gone, CPU becomes bad. If the machine stops bursting, credits are accumulated over time. If our instance consistently runs low on credit, we need to move to a different kind of non-burstable instance like C or M. We can see the credit usage and balance in cloudwatch.
* T2/T3-unlimited: Provides unlimited burst. We pay extra if we go over credit balance.
* Refer: ect2instances.info

AMI (Amazon Machine Image):

* An image to use to create our instances
* We can create custom AMIs
* AMIs are built for specific AWS region
* We can use/rent public AMIs from Market Place, be cautious of malware/security-risks.
* AMIs live in S3, but we won’t see them in S3 console.
* By default, AMIs are private and locked for our account/region. But, we can make them public and share/sell them.
* We get charged for the actual space taken on S3.
* Right-click on instance, choose ‘Image’ -> ‘Create Image’
* Once image is created, right-click and Launch to create a new instance using the AMI
* If we copy an AMI that has been shared with our account, we are the owner of the target AMI in our account. If we do not want someone to copy our AMI, we may not grant them EBS Snapshot access (or) S3 Bucket access. But, still, if someone is able to create an instance from an AMI they will be able to make a copy of that AMI
* If an encrypted AMI is shared with you, once you have the encryption key, you can copy the snapshot while re-encrypting it with a key of your own. You own the copied snapshot and can register it as new AMI.
* Similarly, we can’t copy an AMI with an associated billingProduct code that was shared with us from another account. This includes Windows AMIs and AMIs from AWS marketplace. To copy a shared AMI with a billingProduct code, launch an EC2 instance in your account using the shared AMI and then create an AMI from the instance.
* To share an AMI, right-click, choose ‘Modify Image Permissions’, can choose ‘public’/’private’, and add AWS account-number for ‘private’. If copy permissions are given, the receiver can right-click and make a copy of that AMI in the desired region.

EC2 Placement Groups:

* Provides us control over the EC2 instance placement strategy
* Cluster – Clusters instances into a low latency group in a single AZ. On same Rack/HW and same AZ. If rack fails, all instances fail, but provides great network/speed (like 10Gbps). Used for big-data jobs, Apps that need low-latency.
* Spread – Spreads instances across underlying HW. All instances will be located on different HW, spans across multiple AZ. Used for high-availability needs. There is a limitation of max 7 instances per group per AZ, so our App can’t be too big.
* Partition – Spreads instances across many different partitions within an AZ. Partitions are sets of racks within an AZ. A partition failure can effect many instances but won’t affect other partitions. EC2 instances has access to partition information as metadata. Can have upto 7 partitions per AZ, and can have upto 100s of instances as part of that placement group. Used for distributed big-data applications HDFS, HBase, Cassandra, Kafka
* During instance creation, in Step3, we can choose the placement group for the instance. If ‘partition’ group is selected, can choose the partition number.

ENI (Elastic Network Interfaces):

* ENI is a logical component in a VPC, that represent a virtual network card/interface. They give EC2 instances access to network.
* ENI can have the following attributes:
* Has one Primary private IPv4, Zero or more secondary private IPV4 (e.g. Eth0–primary ENI–192.168.0.31, Eth1-secondary ENI-192.168.0.42)
* Can have one Elastic IP (IPv4) per private IPv4
* Has one public IPv4
* Has one or more security groups attached
* Has MAC address attached to it
* We can create ENI independently and attach them on the fly on EC2 instances for failover. Can move primary/secondary ENIs from one EC2 instance to another.
* Bound to a specific AZ
* For an EC2 instance, we could see in the details that eth0 would be mapped as Network Interface, which is primary. When we see the details of eth0, we see Interface-ID which represents the ENI id.
* In AWS console, can access ENIs, by traversing ‘Network & Security’ -> ‘Network Interfaces’. We can create new network interfaces using this page. Once created, we can right-click and choose ‘attach’ to attach it to any EC2 instance. Also, can add more IP-addresses to a network interface, need to right-click and choose ‘manage IP addresses’ and click on ‘Assign new IP’.

EC2 Hibernate:

On first EC2 start, OS boots & EC2 user data script is run. On subsequent starts, OS boots up, application starts and caches get warmed up