**Commands for Serverless**

**Deploy packages on serverless**

serverless deploy --aws-profile devProfile

Serverless deploy function -f <function name>

**Configuration of aws credentials in serverless**

serverless config credentials --provider aws --key AKIARJEW5RSIV4IYTBNA --secret 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 --profile serverless-admin

**Creating template for python code in serverless**

Serverless create -t --aws-python3

**Invoking the serverless function from serverless framework**

Serverless invoke -f <function name> -l

**Logs**

Monitoring section on lambda page -> check the cloudwatch logs as well

programmatic from console

Serverless logs -f hello -t(tail) -

**Destroy the service-need to cleanup**

* Function
* Dependencies of the function
* Cloudwatch log groups
* Iam roles
* Everything else the framework has created

Serverless remove all of them

Serverless remove

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

**YAML**

* Markup language used to edit 'serverless.yml' file
* Made of key value pairs
* Nested objects
* Support arrays (starts with a minus sign)
* Multi line string (>2 line string)
* '---' at start to indicate the starting point
* Comments starts with '>'

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

**Functions timeout and memory**

Update the serverless.yml file (functions tab)

Hello-short-timeout:

Handler: handler.hello

Timeout: 3(sec)

memorySize: 128

Hello-long-timeout:

Handler: handler.hello

Timeout: 6(sec)

Memorysize: 256

* Statements provided in the provider section applies for all; but if specified in the function itself will override the provider settings.

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

**IAM permissions for lambda functions**

* Lambda function access other services e.g. S3, dynamoDB
* By default lambda functions are not authorized to do that
* Iam allows you to entirely secure your aws setup
* Add iam roles in the serverless yaml files as a json mentioned in the iam itself

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

**Environment variables**

* Environment variables are good because they provide external configuration to your functions
* This way, we can change a functions behavior without even changing the code of the function.
* Pass the variables to be used in the handler code as part of the yaml file in the environment variable section; if environment is specified within the function; it will override the one mentioned above.
* Usecase - dev, prod functions, variables.

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

**VPC for lambda functions**

* VPC are virtual private cloud

Many companies use VPC to privately deploy their applications by default lambda functions are not launched in a VPC

* But you can launch Lambda in your VPC, so that

Your lambda function are securely access to EC2 instances, RDS instances or any other instance

* You can also assign security groups to your lambda functions as well for enhanced network security
* You can choose to deploy your lambda function in any subnets you like
* This allows lambda functions to inherit a private IP from that subnet

e.g.

In the yaml file:

Vpc:

securityGroups:

- securityGroup1

- securityGroup2

subnetIds:

- subnetId1

- subnetId2

* Can be setup at the function level as well which will override in the provider section.

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

**Plugins:**

Create requirements.txt file with all the external packages to be imported into python code; if they are already not a part of python stack. e.g. serverless-python-requirements package and it uses docker and the following code is required in the custom section.

Pythonrequirement:

dockerizePip: true

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

**Real world examples:**

* REST API (http: api gateways) Insonia(rest api desktop app)
* S3 trigger
* EC2 start stop
* Use cron to schedule jobs in the events tab