Cloud Native Application - Serverless I

Cloud Infrastructure Engineering

Nanyang Technological University & Skills Union - 2022/2023

Course Content

- Self Study Check In
- Introduction to Serverless
- Pros & Cons of Serverless
- Build Serverless Applications - Serverless Framework, SAM
- Activity

Activity

Instructor

- Ask to use AWS use single region for all learner for easier monitoring

Q1: What are the serverless computing services provided by AWS?

https://aws.amazon.com/serverless/

Popular Serverless Computing Services

AWS Lambda

Google Cloud Functions

Microsoft Azure Functions

IBM/Apache's OpenWhisk

Oracle Cloud Fn

Q2: When does the serverless model provide the most economic benefit(select two answers)?

1. When the application is event driven
2. When the application is used internally and not externally
3. When event loads are consistent
4. When event loads are inconsistent

Q3: Is Serverless always the best solution?

- Yes
- No

What is Cloud Native?

“Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds.”

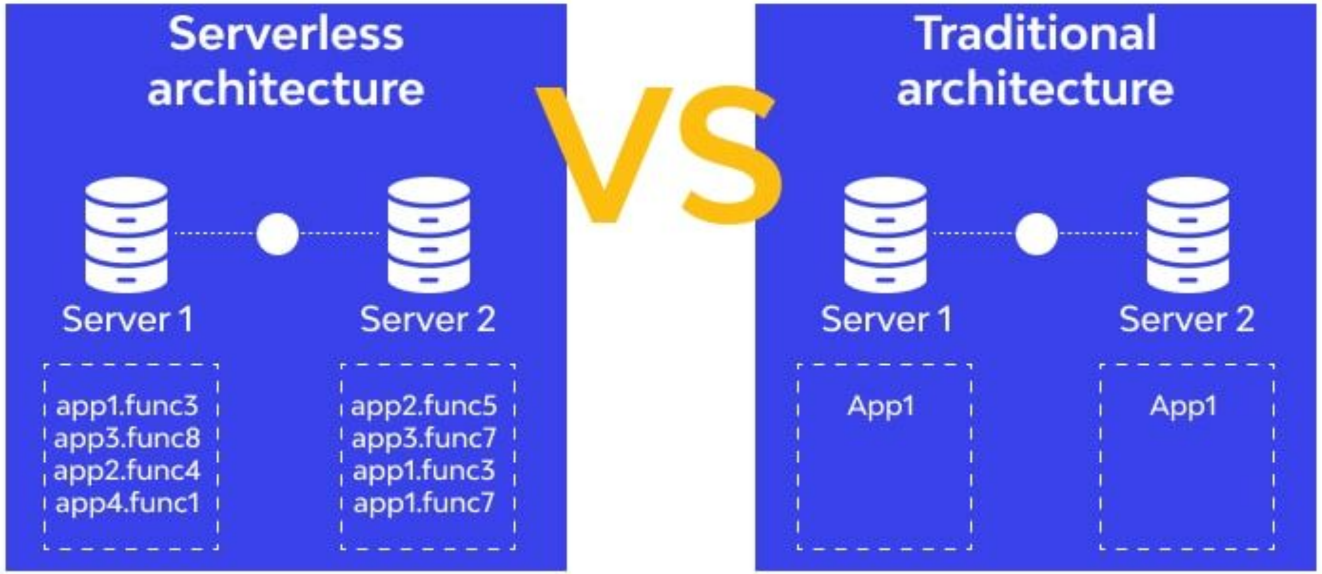
Cloud-native app development typically includes marrying **microservices**, **cloud platforms**, **containers**, **Kubernetes**, **immutable infrastructure**, **declarative APIs**, and **continuous delivery** technology with techniques like **devops** and **agile** methodology.

(Cloud Native Computing Foundation)

Serverless?

Serverless = No Server ?

Serverless VS Traditional Architecture



What is Serverless?

Serverless is a **cloud-native model** where the cloud provider fully manages the underlying server infrastructure

Developers build and **run code without managing servers**

No need to pay for idle cloud infrastructure.

Applications are **broken up into individual functions** that **can be invoked** and **scaled individually**.

Serverless does not mean 'no servers'

The name notwithstanding, there are most definitely servers in serverless computing. 'Serverless' describes the developer's experience with those servers — **servers** are **invisible** to the developer, who doesn't see them, manage them, or interact with them in any way.

You only need to worry about your code.

Use cases for serverless

- Microservices
- API backends
- Asynchronous processing
- Event Driven
- Web Applications
- IT Automation
- Data Processing

Pros of Serverless

- Improve developer productivity
- Pay for execution only
- Develop in any language
- Cost-effective performance
- Streamlined development/ DevOps cycles
- Usage visibility

Cons of Serverless

Unacceptable latency for certain applications

Higher costs for stable or predictable workloads

Monitoring and debugging issues

Privacy and Security Concerns

Serverless in AWS

Computing:

- AWS Lambda
- AWS Fargate

Streaming

- Kinesis Data Streams
- Kinesis Data Firehose

Serverless in AWS

Application Integration:

- AWS Cognito
- AWS API Gateway
- AWS SNS
- AWS SQS
- AWS AppSync
- Step Functions
- AWS Eventbridge

Serverless in AWS

DataStores:

- DynamoDB
- Aurora Serverless
- S3
- EFS
- Redshift Serverless
- Neptune

Tools to build Serverless Applications faster

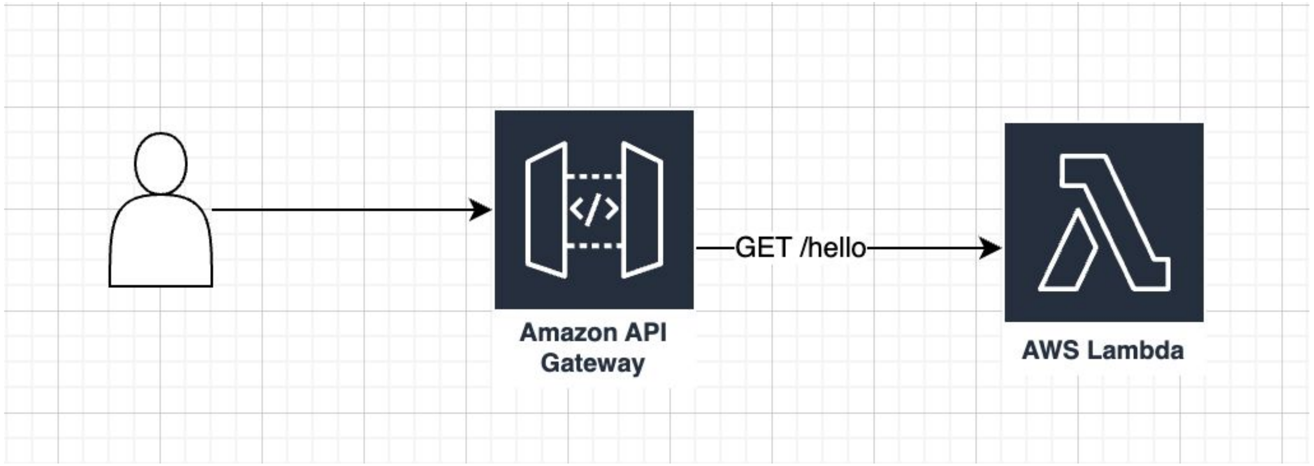
- Serverless Framework
- Serverless Application Model (SAM)
- Chalice
- CDK

serverless

AWS SAM(Serverless Application Model)

Activity

Activity



serverless

What is Serverless Framework?

- **Less Code** - Build more and manage less with serverless architectures.
- **Many Use-Cases** - Choose from tons of efficient serverless use-cases (APIs, Scheduled Tasks, Event Handlers, Streaming Data Pipelines, Web Sockets & more).
- **Automated** - Deploys both code and infrastructure together, resulting in out-of-the-box serverless apps.
- **Easy** - Enjoy simple syntax to safely deploy deploy AWS Lambda functions, event sources and more

  Multi-Language - Supports Node.js, Python, Java, Go, C#, Ruby, Swift, Kotlin, PHP, Scala, & F#
- **Full Lifecycle** - Manages the lifecycle of your serverless architecture (build, deploy, update, monitor, troubleshoot).
- **Multi-Environments** - Built-in support for multiple stages (e.g. development, staging, production).
- **Extensible** - Extend or modify the Framework and its operations via Plugins.

Serverless Framework

What you need to know?

- template specification: **serverless.yml** file
- serverless **cli**
  - create project ⇒ `serverless(or sls)`
  - deploy project ⇒ `sls deploy`

Commands

```
$ npm install serverless -g
```
-> To install serverless globally

```
$ serverless
```

rename the service under serverless.yml

```
$ serverless deploy
```

Command

$ npm init

copy index.js & serverless.yml to the prev. github repo you created.

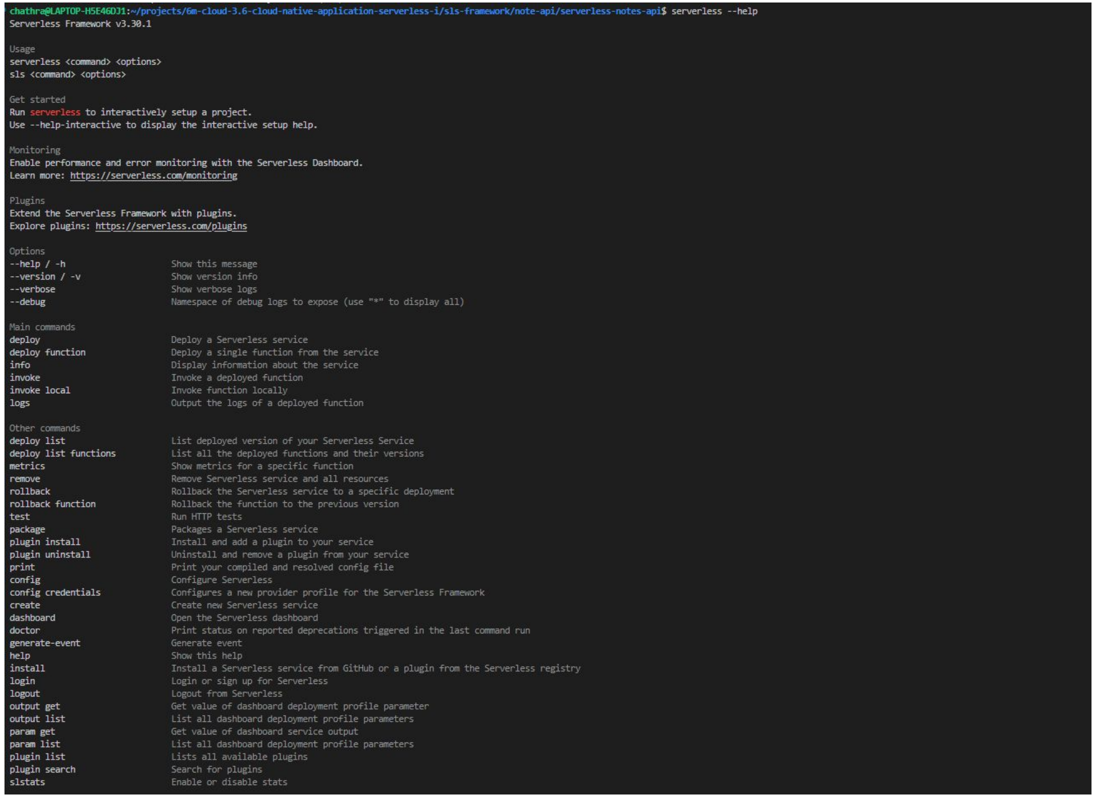
$ npm install serverless-offline --save-dev

add serverless-offline plugin in serverless.yml

```
plugins:
 - serverless-offline
```

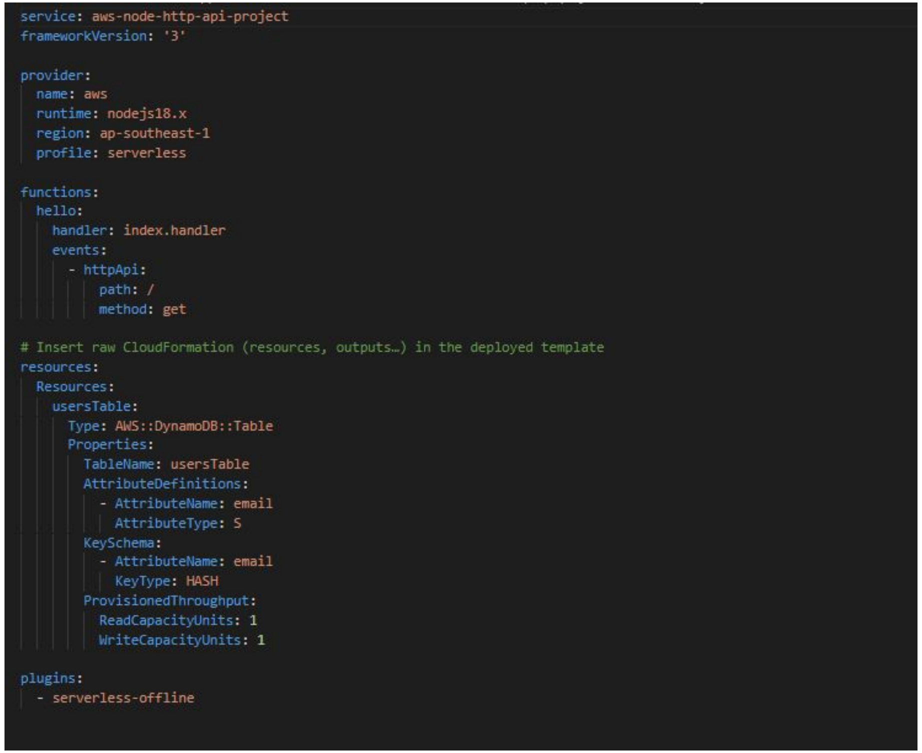
Serverless Framework CLI

- All you need to know ;-) : `serverless --help`



```
chathra@LAPTOP-H5E46DJ1:~/projects/6m-cloud-3.6-cloud-native-application-serverless-i/sls-framework/note-api/serverless-notes-api$ serverless --help
Serverless Framework v3.30.1

Usage
serverless <command> <options>
sls <command> <options>

Get started
Run serverless to interactively setup a project.
Use --help-interactive to display the interactive setup help.

Monitoring
Enable performance and error monitoring with the Serverless Dashboard.
Learn more: https://serverless.com/monitoring

Plugins
Extend the Serverless Framework with plugins.
Explore plugins: https://serverless.com/plugins

Options
--help / -h                     Show this message
--version / -v                  Show version info
--verbose                       Show verbose logs
--debug                         Namespace of debug logs to expose (use "*" to display all)

Main commands
deploy                          Deploy a Serverless service
deploy function                 Deploy a single function from the service
info                            Display information about the service
invoke                          Invoke a deployed function
invoke local                    Invoke function locally
logs                            Output the logs of a deployed function

Other commands
deploy list                     List deployed version of your Serverless Service
deploy list functions           List all the deployed functions and their versions
metrics                         Show metrics for a specific function
remove                          Remove Serverless service and all resources
rollback                        Rollback the Serverless service to a specific deployment
rollback function               Rollback the function to the previous version
test                            Run HTTP tests
package                         Packages a Serverless service
plugin install                  Install and add a plugin to your service
plugin uninstall                Uninstall and remove a plugin from your service
print                           Print your compiled and resolved config file
config                          Configure Serverless
config credentials              Configures a new provider profile for the Serverless Framework
create                          Create new Serverless service
dashboard                       Open the Serverless dashboard
doctor                          Print status on reported deprecations triggered in the last command run
generate-event                  Generate event
help                            Show this help
install                         Install a Serverless service from GitHub or a plugin from the Serverless registry
login                           Login or sign up for Serverless
logout                          Logout from Serverless
output get                      Get value of dashboard deployment profile parameter
output list                     List all dashboard deployment profile parameters
param get                       Get value of dashboard service output
param list                      List all dashboard deployment profile parameters
plugin list                     Lists all available plugins
plugin search                   Search for plugins
slstats                         Enable or disable stats
```

Serverless Framework - serverless.yml

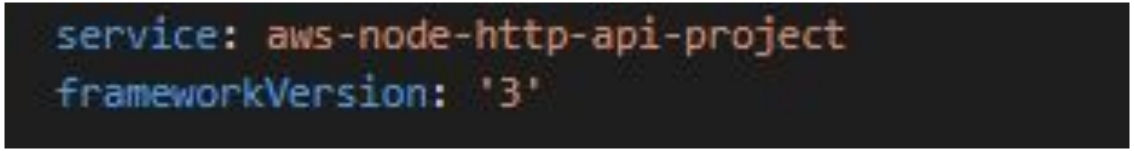


```yaml
service: aws-node-http-api-project
frameworkVersion: '3'

provider:
  name: aws
  runtime: nodejs18.x
  region: ap-southeast-1
  profile: serverless

functions:
  hello:
    handler: index.handler
    events:
      - httpApi:
          path: /
          method: get

# Insert raw CloudFormation (resources, outputs…) in the deployed template
resources:
  Resources:
    usersTable:
      Type: AWS::DynamoDB::Table
      Properties:
        TableName: usersTable
        AttributeDefinitions:
          - AttributeName: email
            AttributeType: S
        KeySchema:
          - AttributeName: email
            KeyType: HASH
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1

plugins:
  - serverless-offline
```

Serverless Framework Concepts

Services

- A service is the Framework's unit of organization. You can think of it as a project file, though you can have multiple services for a single application.



```
service: aws-node-http-api-project
frameworkVersion: '3'
```

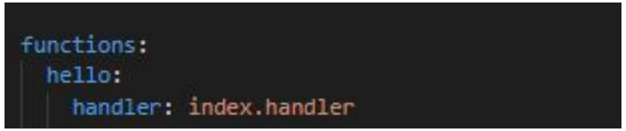
Serverless Framework Concepts

Functions

The code of a serverless application is deployed and executed in **AWS Lambda functions**.

Each function is an independent unit of execution and deployment, like a microservice. A function is merely code, deployed in the cloud, that is most often written to perform a single job such as:

- Saving a user to the database
- Processing a file in a database
- Performing a scheduled task



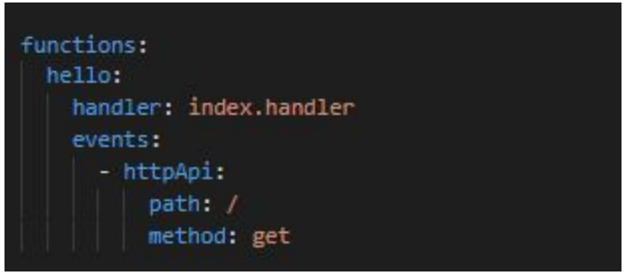
```
functions:
  hello:
    handler: index.handler
```

Serverless Framework Concepts

Events

Functions are **triggered** by events. Events come from other AWS resources, for example:

- An HTTP request on an **API Gateway** URL (e.g. for a REST API)
- **S3** events (e.g. for an image upload)
- A **CloudWatch schedule** (e.g. run every 5 minutes)
- A message in an **SNS** topic
- A **CloudWatch alert**
- And more...



```yaml
functions:
  hello:
    handler: index.handler
    events:
      - httpApi:
          path: /
          method: get
```

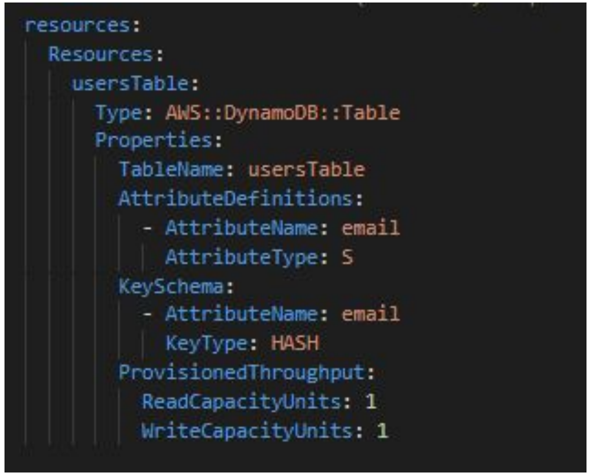
When you configure an event on a Lambda function, Serverless Framework will automatically create the infrastructure needed for that event (e.g. an API Gateway endpoint) and configure your functions to listen to it.

Serverless Framework Concepts

Resources

Resources are AWS infrastructure components which your functions use such as:

- A DynamoDB table (e.g. for saving users/posts/comments data)
- An S3 Bucket (e.g. for saving images or files)
- An SNS topic (e.g. for sending messages asynchronously) ….



```
resources:
  Resources:
    usersTable:
      Type: AWS::DynamoDB::Table
      Properties:
        TableName: usersTable
        AttributeDefinitions:
          - AttributeName: email
            AttributeType: S
        KeySchema:
          - AttributeName: email
            KeyType: HASH
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
```

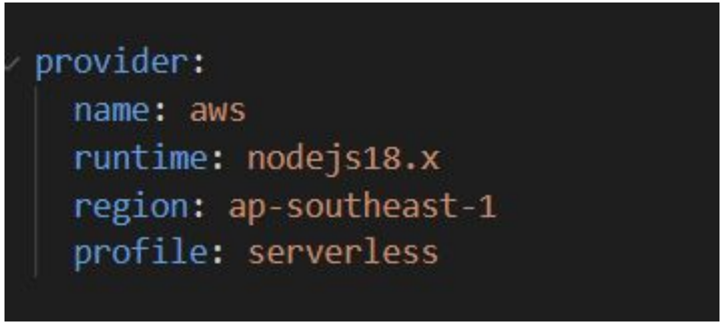
Anything that can be defined in **CloudFormation** is supported by the Serverless Framework

Serverless Framework can deploy functions and their events, but also AWS resources.

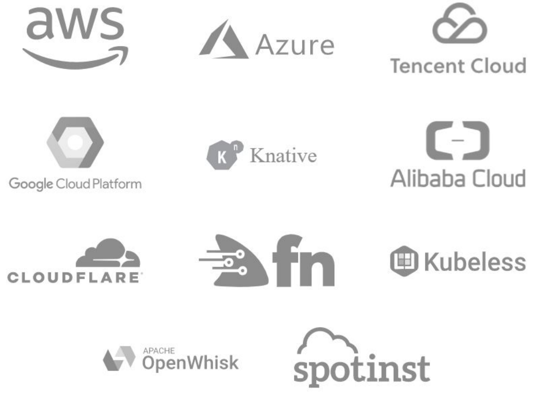
Serverless Framework Concepts

Provider

To deploy functions, specify your provider in your service's serverless.yml file under the provider key and make sure your provider credentials are setup on your machine or CI/CD system.



```yaml
provider:
  name: aws
  runtime: nodejs18.x
  region: ap-southeast-1
  profile: serverless
```



AWS SAM(Serverless Application Model)

What is Serverless Application Model(SAM)?

- **Built on AWS CloudFormation –** Use the AWS CloudFormation syntax directly within your AWS SAM template, taking advantage of its extensive support of resource and property configurations. If you are already familiar with AWS CloudFormation, you don't have to learn a new service to manage your application infrastructure code.
- **An extension of AWS CloudFormation –** AWS SAM offers its own unique syntax that focuses specifically on speeding up serverless development. You can use both the AWS CloudFormation and AWS SAM syntax within the same template.
- **An abstract, shorthand syntax –** Using the AWS SAM syntax, you can define your infrastructure quickly, in fewer lines of code, and with a lower chance of errors. Its syntax is especially curated to abstract away the complexity in defining your serverless application infrastructure.
- **Transformational –** AWS SAM does the complex work of transforming your template into the code necessary to provision your infrastructure through AWS CloudFormation.

Serverless Application Model(SAM)

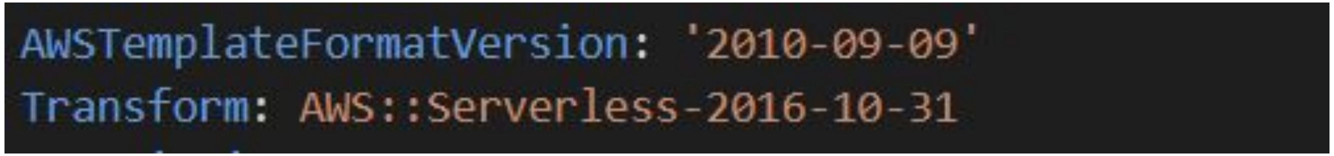
What you need to know?

- Template specification: **eg:- template.yaml** file(YAML or JSON)
- sam **cli**
  - create project ⇒ `sam init`
  - build project ⇒ `sam build`
  - deploy project ⇒ `sam deploy`
- sam **cli config file - samconfig.toml**

Serverless Application Model(SAM) -Template

Transform declaration

- The declaration `Transform: AWS::Serverless-2016-10-31` is required for AWS SAM template files.
- This declaration identifies an AWS CloudFormation template file as an AWS SAM template file.

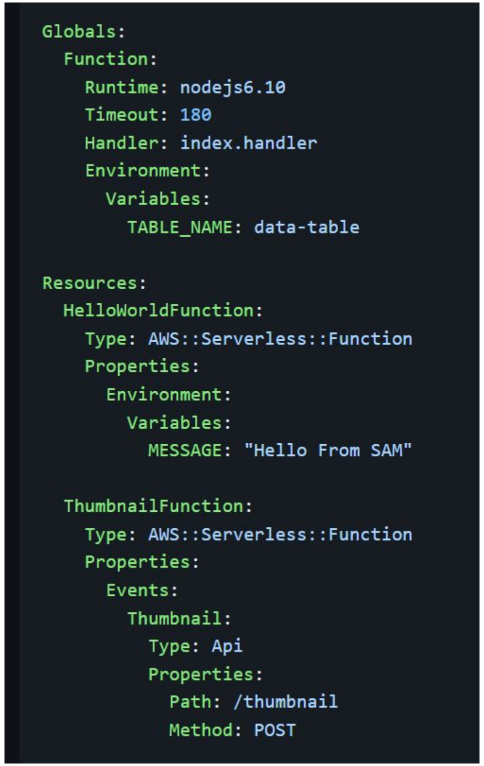


```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
```

Serverless Application Model(SAM) -Template

Globals section

- Unique to AWS SAM.
- It defines properties that are common to all your serverless functions and APIs.
- Resources in a SAM template tend to have shared configuration such as Runtime, Memory,
- VPC Settings, Environment Variables, CORS, etc
- Instead of duplicating this information in every resource, you can write them once in the Globals section and let all resources inherit it.



```yaml
Globals:
  Function:
    Runtime: nodejs6.10
    Timeout: 180
    Handler: index.handler
    Environment:
      Variables:
        TABLE_NAME: data-table

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          MESSAGE: "Hello From SAM"

  ThumbnailFunction:
    Type: AWS::Serverless::Function
    Properties:
      Events:
        Thumbnail:
          Type: Api
          Properties:
            Path: /thumbnail
            Method: POST
```

Serverless Application Model(SAM) -Template

Resources section.

In AWS SAM templates the `Resources` section can contain a combination of AWS CloudFormation resources and AWS SAM resources.

- Cloudformation: AWS resource and property types reference
- SAM: AWS SAM resource and property reference.

Serverless Application Model(SAM) -Template

Parameters section(optional)

Objects that are declared in the `Parameters` section cause the `sam deploy --guided` command to present additional prompts to the user.
Once you run the `sam deploy --guided` command it will create a samconfig.toml which includes all parameters.

For examples of declared objects and the corresponding prompts, see sam deploy in the AWS SAM CLI command reference.

Serverless Application Model(SAM) -Template

CloudFormation Resources Generated By SAM

https://github.com/aws/serverless-application-model/blob/master/docs/internals/generated_resources.rst#cloudformation-resources-generated-by-sam

Serverless Application Model(SAM)

SAM CLI Configuration file(samconfig.toml)

The AWS SAM CLI supports a project-level configuration file that stores default parameters for its commands.

This configuration file is in the TOML file format, and the default file name is `samconfig.toml`.

The file's default location is your project's root directory, which contains your project's AWS SAM template file.

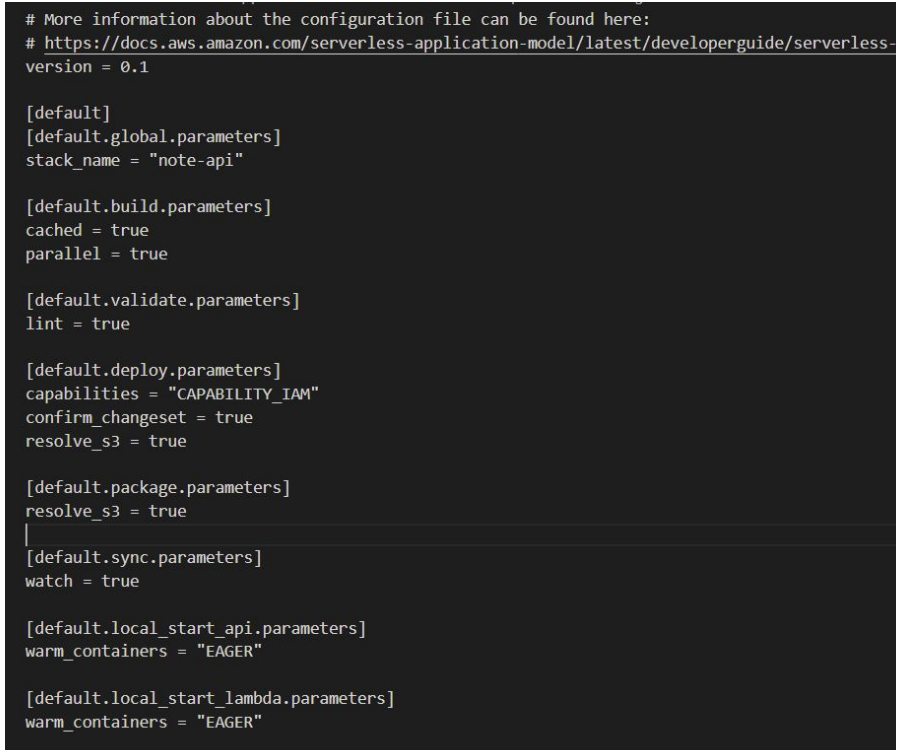
How to set the parameters?

- `sam deploy --guided` command writes a subset of parameters to your configuration file.(Recommended)
- Manually editing the file

  Syntax: For commands, the format of the table header is **[environment.command.parameters].**

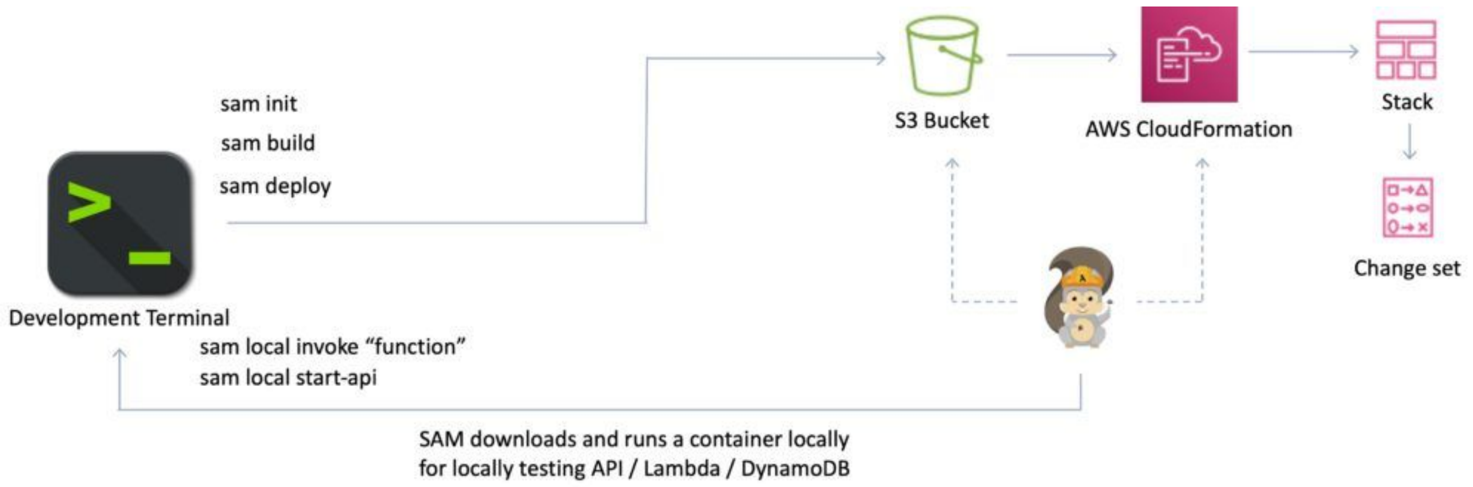
  eg:- for the sam deploy command for the default environment, the configuration table header is **[default.deploy.parameters]**

Serverless Application Model(SAM)



```
# More information about the configuration file can be found here:
# https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-
version = 0.1

[default]
[default.global.parameters]
stack_name = "note-api"

[default.build.parameters]
cached = true
parallel = true

[default.validate.parameters]
lint = true

[default.deploy.parameters]
capabilities = "CAPABILITY_IAM"
confirm_changeset = true
resolve_s3 = true

[default.package.parameters]
resolve_s3 = true

[default.sync.parameters]
watch = true

[default.local_start_api.parameters]
warm_containers = "EAGER"

[default.local_start_lambda.parameters]
warm_containers = "EAGER"
```

Serverless Application Model


sam init
sam build
sam deploy
Development Terminal
S3 Bucket
AWS CloudFormation
Stack
Change set
sam local invoke "function"
sam local start-api
SAM downloads and runs a container locally
for locally testing API / Lambda / DynamoDB

Serverless Application Model

AWS::Serverless::Function - AWS Lambda

AWS::Serverless::Api - API Gateway

AWS::Serverless::SimpleTable - DynamoDB

AWS::Serverless::Application - AWS Serverless Application Repository

AWS::Serverless::HttpApi - API Gateway HTTP API

AWS::Serverless::LayerVersion - Lambda layers

Setting up Serverless CLI in your machine

Let spend 5 - 10 mins to configure and install all these requirements.

- Learner create new repository on github.
- Install serverless locally (**npm install -g serverless**)
- Configure serverless account locally (not mandatory)
- Configure AWS CLI (Check if you done it before)

Documentations:

- https://www.serverless.com/framework/docs/getting-started
- https://www.serverless.com/framework/docs/providers/aws/guide/credentials

Create the First Serverless Demo Application

Instructor Demo First Serverless Application

```
# Create a new serverless project

serverless



# Move into the newly created directory

cd your-service-name
```

Push all new file to github

Learner - 10 mins

Learners try to create First Serverless Application and push all the code to github

Activity

Instructor Invoke Serverless Function

```
serverless invoke -f hello



# Invoke and display logs:

serverless invoke -f hello --log
```

Activity

Learners Invoke Serverless Function

Activity

Instructor Install Serverless Offline

```
npm install serverless-offline --save-dev
```

Add serverless offline on `serverless.yml`

```
plugins:

    - serverless-offline
```

Activity

Learners Install Serverless Offline

Activity

Instructor Remove serverless function

```
serverless remove
```

Activity

Learners Remove serverless function

Activity

Create a new repo in Github and clone it to your local computer

.gitignore template -> Terraform

Add README.md

Activity

Create a lambda.tf file with below content:

https://github.com/jaezeu/terraform-serverless/blob/main/apigateway-lambda/lambda.tf

Activity

Create a lambda_function.py file with below content:

```
import boto3

def lambda_handler(event, context):

    result = "Hello World"

    return {

        'statusCode' : 200,

        'body': result

    }
```

Activity

Create a provider.tf file with below content:

```
provider "aws" {

  region = "ap-southeast-1"

}
```

Activity

Create a variables.tf file with below content:

```
variable "your_name" {
  type = string
  default = "jaz"  #Replace with your name here
}
```

Activity

Once all 6 files above have been created, Run the following commands:

terraform init

terraform plan

terraform apply

Question/Challenge:

- Is it possible to get the api gateway url without going to the console?

Questions?

Activity

Learner:

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.

Instructor

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.
- Check the AWS account after learner clean up.

END