ServerLess Computing - LAMBDA

- **AWS Lambda** – Run code without thinking about server
- **Compute Service:** AWS Lambda is a server-less compute service.
- **Amazon's Infrastructure:** It runs your code in response to events and automatically manages the compute resources for you.
- **Function as a Service:** Automatically run code in response to events (like s3, dynamodb, API gateway, http call etc) which is called triggers in Lambda.
- **Autoscaling and DR:** Runs the code in high-availability compute infrastructure with automatic scaling capabilities.
- **Supported by:** Java, Python, NodeJS

Serverless Computing - LAMBDA

- Your code, is a function in AWS LAMBDA
- Forgot about the servers, its all handled by AWS and you just need to pay for the time your code is running.
- Functions should be stateless, but you have the flexibility to use S3, DynamoDB (other AWS services) to store stateful functions.
- **AWS LAMBDA "Serverless Service", In reality "Still there are servers!"** but managed by AWS with Autoscaling enabled by default.
- Pricing: Free tier is available for testing
  with 1M requests
  3M seconds of compute time
- You are charged for every 100ms your code executes and the number of times your code is triggered. No usage means No cost to pay.

LAB 47 : First LAMBDA function

- Go to Lambda console
- Select hello-world blueprint
- Give Function name and select Node.js
- Select Role (make it simple for the first time)
- Create Function, Save and Run with key values

First LAMBDA function

Code entry type: Edit code inline

```
'use strict';

console.log('Loading function');

exports.handler = (event, context, callback) => {
    //console.log('Received event:', JSON.stringify(event, null, 2));
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, event.key1);  // Echo back the first key value
    //callback('Something went wrong');
};
```

Execution result: succeeded (logs)

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
"hello"
```

First LAMBDA function

- Execution Result
- Summary
- Log output

Lambda - Facts

Throttle/Concurrency:

Throttling is concurrent execution limit control. You can throttle your function both the account level and the function level. Use throttling:

- In case indefinite scaling in backend is not desired
- to have a controlled cost factor
- to regulate how long it takes you to process a batch of events
- to match function limit with a downstream/backend resource i.e. sql concurrent session limits.
- set the concurrency to 0, to stop all invocations (in case require same to hold execution temporarily)
- Handle it with care. If you are reserving the concurrency for one function, rest function will be limited to use the concurrent executions.

Note: By default, 1000 concurrent executions allowed per account, out of which 100 concurrent executions must remain unreserved.

Lambda - Facts

- Versioning of your Function code can be done.
- You can export your function to work locally or save/share with others.
- You can allocate/limit memory to your function (Cost factor associated).
- Function timeout can be set to avoid cost, due to code errors.
- AWS Lambda automatically retry (twice) failed executions for **asynchronous** (event based) invocations.
- AWS Lambda directs events that cannot be processed to the specified Amazon SNS topic or Amazon SQS queue. Functions that don't specify a DLQ will discard events after they have exhausted their retries.
- Cloud trail can be created for Lambda functions invocations logging.

Lambda - Facts

- **Test events can be configured for Lambda functions, to trigger its execution.**

- These events can be an input from Alexa, CloudFront, Code commit, SNS etc.. Refer test events for latest list of supported events.

- Traffic Shifting Using Aliases:
  - You can create aliases to shift your traffic to two different versions simultaneously.
  - You can define the traffic percentage weightage for each version.
  - You can watch the result-executed-version in cloud watch logs.
  - Mainly used to check the new version in production, before fully integrating same.

- Invocation errors, Throttled invocations, DLQ errors etc. can be monitored through the monitoring tab. Same should be regularly checked.

LAB 48 : LAMBDA function

Random Number Generator

```
'use strict';
console.log('Loading function');
exports.handler = (event, context, callback) => {

        let min = 0;
        let max = 10;

        let RandomNumberGenerator = Math.floor(Math.random() * max) + min;

        callback(null, RandomNumberGenerator);
};
```

Note: Lambda Function Name and name mentioned in the code should be same (like RandomNumberGenerator) is my Lambda Function name and the same used in the code.

LAB 48 : LAMBDA function

Random Number Generator

Code entry type: Edit code inline

Runtime: Node.js 6.10

Handler Info: index.handler

File Edit Find View Goto Tools Window

Environment

RandomNumberGenerator

index.js

index.js

```
1  'use strict';
2  console.log('Loading function');
3  exports.handler = (event, context, callback) => {
4
5      let min = 0;
6      let max = 10;
7
8      let RandomNumberGenerator = Math.floor(Math.random() * max) + min;
9
10     callback(null, RandomNumberGenerator);
11 };
12
```

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
3
```

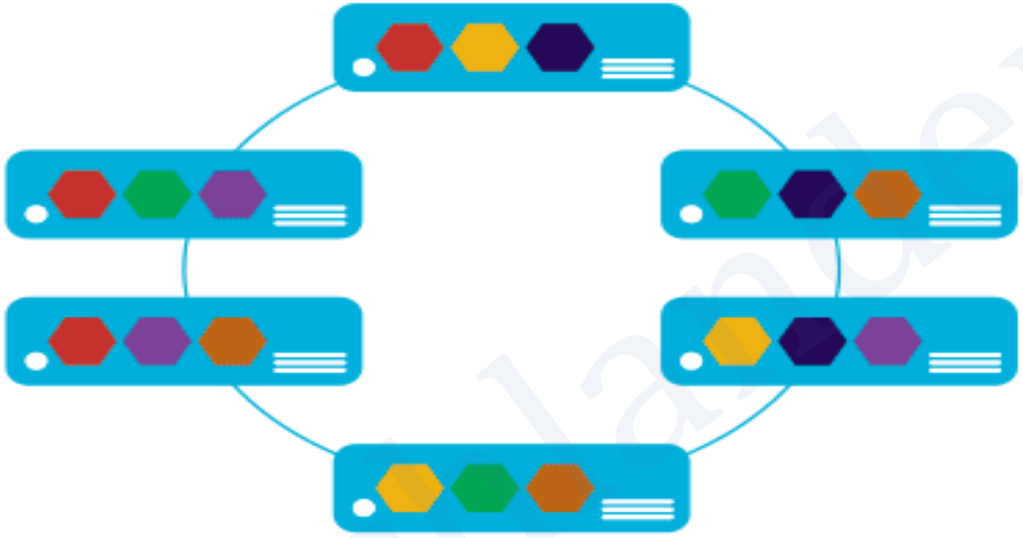
Containers

Micro Services vs Monolithic

Microservices Approach

A microservice approach segregates functionality into small autonomous services.

And scales out by **deploying independently** and replicating these services across servers/VMs/containers.

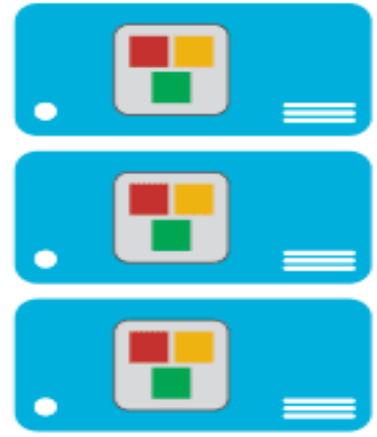


VS.

Traditional Approach

A traditional application (Web app or large service) usually has most of its functionality within a single process (usually internally layered, though).

And scales by cloning the whole app on multiple servers/VMs/containers.



AWS Elastic Container Service

- Amazon Elastic Container Service (Amazon ECS) is the Amazon Web Service you use to run Docker applications on a scalable cluster.
- Amazon ECS makes it easy to deploy, manage, and scale Docker containers running applications, services, and batch processes
- Amazon ECS places containers across your cluster based on your resource needs and is integrated with familiar features like Elastic Load Balancing, EC2 security groups, EBS volumes and IAM roles
- Eliminates the need for you to install, operate, and scale your own cluster management infrastructure.
- Existing ECS Cluster is Blox (Open source project for container management) based cluster.

Containers Facts

To deploy applications on Amazon ECS, your application components must be architected to run in *containers*.

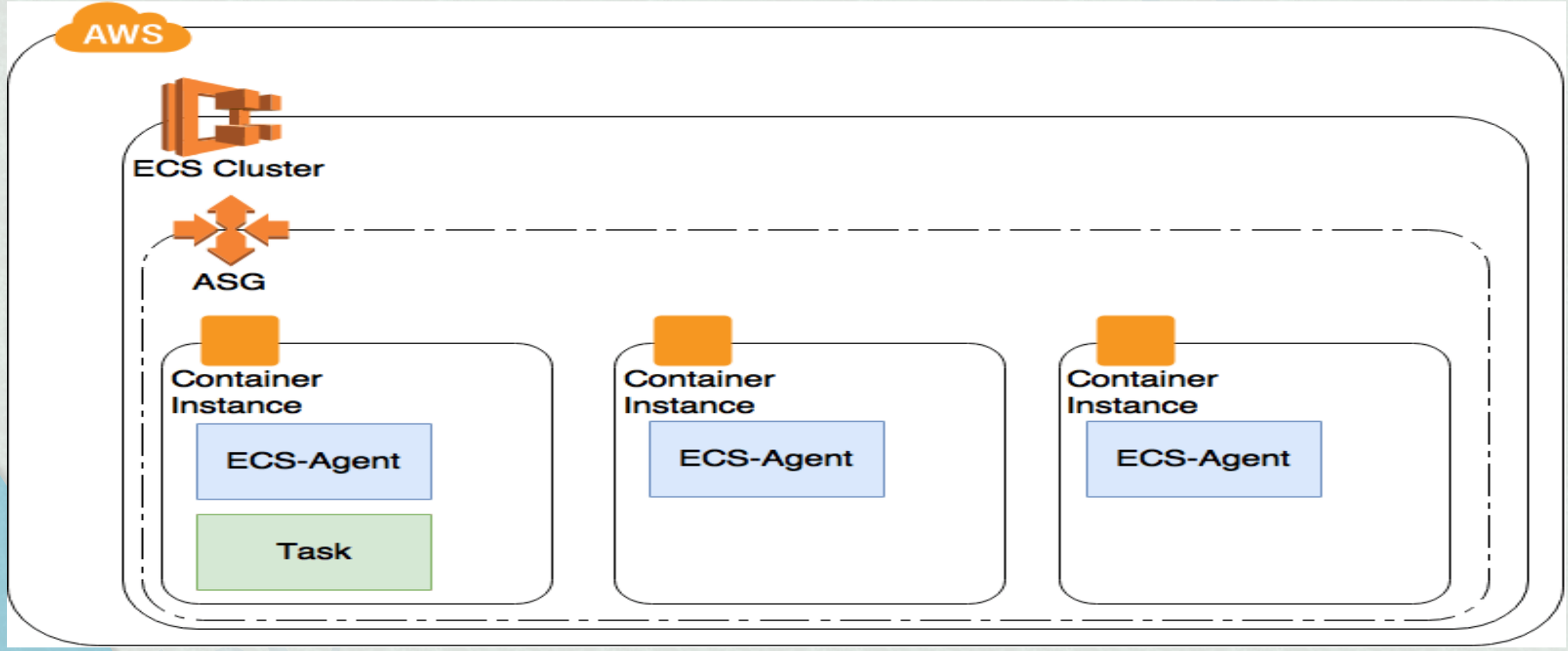
A Docker container is a standardized unit of software development, containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc. Containers are created from a read-only template called an *image*.

Images are typically built from a Dockerfile, a plain text file that specifies all of the components that are included in the container.

AWS Elastic Container Service

- **Facts**
  - No Need to create/manage the cluster environment
  - Autoscaling, Clustering is automatically managed in backend
  - Containers can run on any node in cluster (Across AZ too)
  - Application run here as tasks (similar to pods in Kubernities)
  - Cluster instances are EC2 in backend.
  - ECS is using Autoscaling in Backend (ELB can be configured)

AWS Elastic Container Service



ECS Facts

Containers

- Containers are Light weighted VMs with shared Kernel. Container definition includes, Image name, Port mappings, health checks, Env etc.

Task Definitions

- A *task* is the instantiation of a task definition within a cluster. To prepare your application to run on Amazon ECS, you create a *task definition*. The task definition is a text file, in JSON format, that describes executions roles, one or more containers, up to a maximum of ten, that form your application. It can be thought of as a blueprint for your application.

Services

- Amazon ECS allows you to run and maintain a specified number (the "desired count") of instances of a task definition simultaneously in an Amazon ECS cluster. This is called a service. If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it and maintain the desired count of tasks in the service.

ECS Facts

Cluster

- Logical grouping of resources/Backend instances, which run your Blox cluster behind AS and LB.
- If you use the Fargate launch type with tasks within your cluster, Amazon ECS manages your cluster resources. If you use the EC2 launch type, then your clusters will be a group of container instances you manage.

Container Agent

- The *container agent* runs on each infrastructure resource within an Amazon ECS cluster. It sends information about the resource's current running tasks and resource utilization to Amazon ECS, and starts and stops tasks whenever it receives a request from Amazon ECS.

Amazon Elastic Container Registry

- Amazon ECR is a managed AWS Docker registry service that is secure, scalable, and reliable. Amazon ECR supports private Docker repositories with resource-based permissions using IAM so that specific users or EC2 instances can access repositories and images.

ECR (Elastic Container Registry)

Amazon Elastic Container Registry (ECR) is a fully-managed container registry that makes it easy for developers to store, manage, and deploy container images

- Fully managed
- Highly available
- Secure
- integrates with Amazon ECS and the Docker CLI,

LAB 49 : ECS

Create a ECS Cluster:

In this exercise, you will create an AWS ECS Cluster:

1. Log in to the AWS Management Console and click on ECS → "Create Cluster"
2. Select "EC2 Linux + Networking" under cluster template
3. Select **on demand instances**,
   instance type **T2-micro**,
   number of instance -**2**,
   keypair,
   VPC and subnets
4. Click create.
5. Your ECS cluster is ready in few minutes

LAB 50 : ECS

Create a ECS task defination:

In this exercise, you will create an Task Definition:

1. Log in to the AWS Management Console and click on ECS → “Create task definition”
2. Provide “Task Definition Name” under cluster template
3. Configure task role,
   Network type Bridge,
   Memory 300, cpu 1 vcpu,
4. Add container → Provide container details
5. Click create.
6. Your task definition is ready

LAB 58 : ECS

Check containers:

In this exercise, you will create ECS Services

- Login to Servers and check your containers, running on it

  docker version

  Docker ps

  docker pull centos

  docker images

  Docker stop [instance-id]

  docker exec -it edbd7c16d6ff bash

  Ctlr + p +q //to exit from container

- Observe that post stopping the container, your cluster service is starting same again.

AWS Fargate

- AWS Fargate is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of EC2 instances.
- When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application.
- You pay for the amount of vCPU and memory resources that your containerized application requests.
- Requires Network mode to be awsvpc.
- Fargate supports only Docker Public Repo or AWS Repo. No outside private Repo is allowed for now.
- Currently available in us-east-1, us-east-2, eu-west-1, us-west-2 Regions.

AWS ECS for Kubernetes

- Amazon Elastic Container Service for Kubernetes (Amazon EKS) is a managed service that makes it easy for you to run Kubernetes on AWS without needing to install and operate your own Kubernetes clusters.

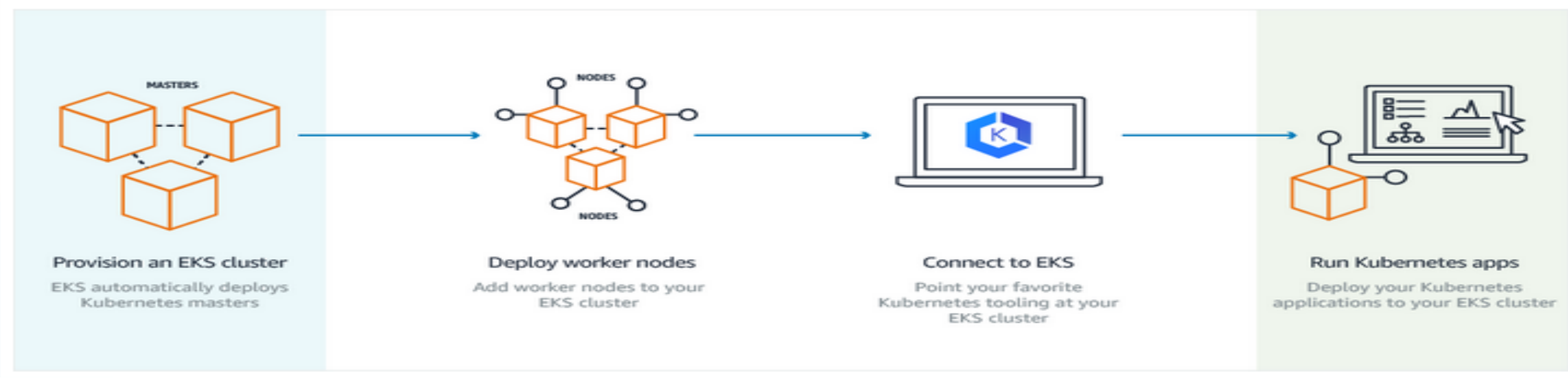
- With Amazon EKS you get a highly-available, and secure Kubernetes control plane without needing to worry about provisioning, upgrades, or patching.

- It is certified Kubernetes conformant so you can use all existing plugins and tooling from the Kubernetes community.

- Any application running on any standard Kubernetes environment is fully compatible.

AWS ECS for Kubernetes

- No Masters to manage, no upgrades to take care of.
- Secure by Default
- Conformant and Compatible
- Allow Hybrid Container Deployments
- Application Migration



Common Practical Approaches

Development and test scenarios

Flexibility for Innovation

Load testing and Live Site swapping

Managed Services

Hybrid scenarios

Burst to the cloud

Latest technology within minutes

AWS as DR

Questions & Answers

THANK YOU

http://blog.Techlanders.com/

For any queries or questions, please contact:

support@Techlanders.com
raman.khanna@Techlanders.com