Stackgenie
Moving from Intel to AMD-powered instances

Demonstrating the seamless transition of workloads to more efficient AMD-based instances.

stackgenie

A little change can go a long way, saving you up to 10% of your compute costs

Making a simple change to a single line of code can save AWS customers up to 10% of their compute costs. A perfect solution for those customers looking at price optimised compute options with greater flexibility for right-sizing workloads.

stackgenie

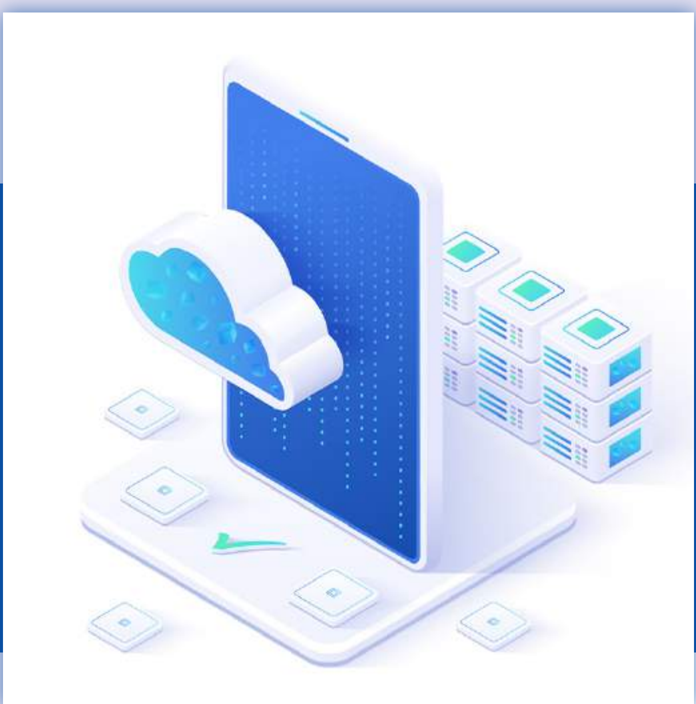
TABLE OF CONTENTS

INTRODUCTION

As a cloud service, Amazon Elastic Compute Cloud (Amazon EC2) provides its customers with cloud-based resizable compute capacity. Allowing users to have complete control of their computing resources whilst running on Amazon's proven computing environment.

Utilising cloud-computing through Amazon EC2, means developers can create and launch new server instances in mere minutes, thus allowing quick scaling capabilities through elastic web-scale computing.

Intel has a long-standing relationship, in excess of 15 years, with Amazon Web Services (AWS) collaborating on developing, building and supporting cloud services. This partnership has made technology more accessible and allowed AWS customers to push the boundaries of innovation.

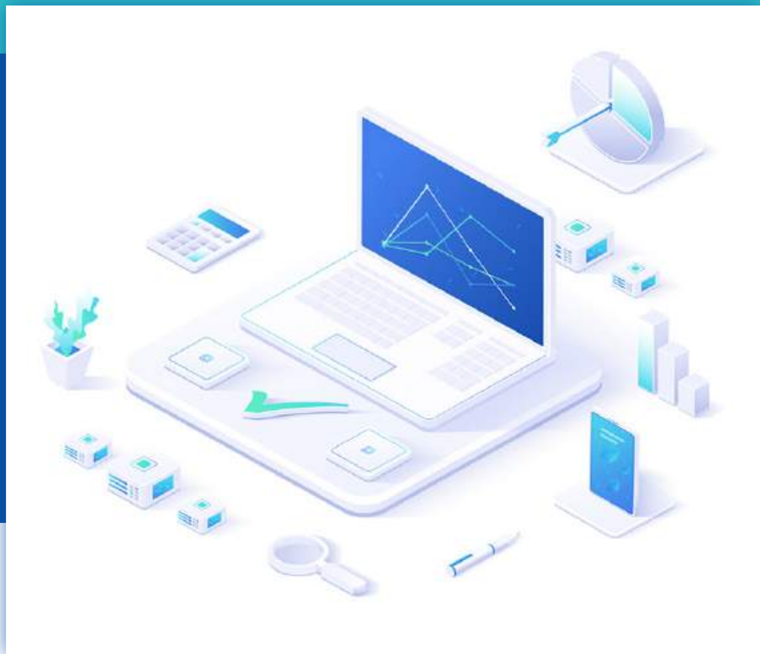


However, with the exponential growth of workloads, each with different characteristics and infrastructure needs, AWS recognised its customers had limited choices for running workloads that were also optimised for performance and cost.

Partnering with AMD since 2018, AWS now delivers a wide variety of choices to right-size workloads whilst simultaneously lowering compute and memory costs for its customers.

The AWS and AMD collaboration resulted in the first generation AMD EPYC™ processors in 2018, followed by the second generation version in 2020, and more recently combining the second generation AMD EPYC™ processors and AMD Radeon Pro GPUs with Amazon EC2 G4ad instances. The launch of third generation AMD EPYC™ processors will further increase the flexibility and choices available too.

AWS customers can use EC2 instances powered by AMD for a wide variety of workloads including databases, enterprise applications, big data analytics, batch processing and gaming.



AMD EPYC™ BENEFITS

Flexibility and Choice

Originally built to provide AWS customers more choice when running Amazon EC2 instances using AMD EPYC™ processors. AMD-powered instances provide flexibility and choice through helping optimise both cost and performance of workloads.

Customers can improve this optimisation further by incorporating right-sizing during the transition process, as well as periodically reviewing as an ongoing process within their organisation.

Cost-savings

Not only do EC2 instances that feature AMD EPYC™ processors deliver up to 10% lower costs for Worldwide regions than comparable instances, the C5a instances also offers the lowest price per x86 vCPU for EC2-based workloads too.

Additionally, in the Asia Pacific (Mumbai) region, EC2 customers are seeing up to 45% lower costs than comparable instances when using AMD EPYC™ processors.

Seamless Workload Transition

For applications running on Amazon's existing x86 EC2 instances (powered by Intel), customers can easily migrate over to the AMD variants with minimal, if any, modification requirements. Switch from C5, T3, M5 and R5 instances to the AMD variants that are available in the same sizes and offer application compatibility.

WORKLOADS COMPARISON

Manufacture	AMD EPYC™	Intel
General Purpose	**T3a** Unlimited CPU burst 0.5:2 GiB to vCPU Up to 2.5 GHz 1st Gen AMD EPYC* Up to 8vCPU / 32 GiB	**T3** Burstable CPU usage SKX-up to 8 vCPUs
	M5a 4:1 GiB to vCPU Up to 2.5 GHz 1st Gen AMD EPYC* Up to 20 Gbps network Up to 96 vCPU / 384 GiB	**M5** Non-burstable CPU usage SKX-up to 96 vCPUs
Compute Optimised	**C5a** Up to 3.3 GHz EPYC* (2nd Gen) Up to 25 Gbps network Up to 96 vCPU / 192 GiB	**C5** High-performance low price/ compute ratio SKX-up to 36vCPUs
Memory Optimised	**R5a** | 8:1 GiB to vCPU Up to 2.5 GHz 1st Gen AMD EPYC* Up to 20 Gbps network Up to 768 GiB / 96 vCPU	**R5** | up to 768 GiB RAM SKX or CLX Up to 96 VCPUs
Graphics-intensive	**G4ad** Ultra-Advanced Up to 64 vCPUs Memory: up to 256 GPU: 32 AMD Radeon Pro V520 Storage: up to 2400	**G4dn** Advanced Up to 96 vCPUs Memory: upto 384 GPU: up to 128 NVIDIA T4 Tensor Storage: up to 2x900

On-demand instances prices

MOVING FROM INTEL TO AMD EPYC™ VARIANTS

Amazon Elastic Compute Cloud (EC2) offers the broadest and deepest compute platform, with over 400 instances and choice of processor, storage, networking, operating system, and purchase model.

EC2 allows users to build apps to automate scaling according to changing needs and peak periods, and makes it simple to deploy virtual servers and manage storage, lessening the need to invest in hardware and helping streamline development processes.

AMD EPYC™ 7000 series processors feature an all core turbo clock speed of 2.5GHz. Amazon EC2 instances powered by AMD EPYC™ processors can deliver optimised compute and memory at a lower cost than comparable instances.

Since many workloads utilise only a fraction of a processor's maximum performance, these instances offer a better fit for purpose for many workloads. Therefore, AMD-based instances provide additional options for AWS customers that are not fully utilising their compute resources, and can result in a cost savings benefit of up to 10% too.

As AMD EPYC™ processors are based on the same x86-64 architecture as Intel processors, applications that are already running on existing EC2 instances can easily be transitioned across. In most cases, with minimal, if any, modification requirements. Due to the application compatibility for R5, M5, T3 and C5 instances, the transition to AMD EPYC™ variants is as simple as stopping an instance, switching the type to AMD and starting it back up.

05

SIMPLE EC2 INSTANCE UPDATE

Update from AWS Console

This section demonstrates how an application running in an AWS EC2 Intel-based instance can be moved to an EC2 AMD-based instance using the web application, AWS Management Console, which comprises of, and refers to, a broad collection of service consoles for managing Amazon Web Services.

The Process

01

Verify the current instance type.

C6g C6gn C6i C5 C5a C5n C4

C5 instances are optimized for compute-intensive workloads and deliver cost-effective high performance at a low price per compute ratio.

Features:

- C5 instances offer a choice of processors based on the size of the instance.
- New C5 and C5d 12xlarge, 24xlarge, and metal instance sizes feature custom 2nd generation Intel Xeon Scalable Processors (Cascade Lake) with a sustained all core Turbo frequency of 3.6GHz and single core turbo frequency of up to 3.9GHz.
- Other C5 instance sizes will launch on the 2nd generation Intel Xeon Scalable Processors (Cascade Lake) or 1st generation Intel Xeon Platinum 8000 series (Skylake-SP) processor with a sustained all core Turbo frequency of up to 3.4GHz, and single core turbo frequency of up to 3.5 GHz.
- New larger 24xlarge instance size offering 96 vCPUs, 192 GiB of memory, and optional 3.6TB local NVMe-based SSDs
- Requires HVM AMIs that include drivers for ENA and NVMe
- With C5d instances, local NVMe-based SSDs are physically connected to the host server and provide block-level storage that is coupled to the lifetime of the C5 instance
- Elastic Network Adapter (ENA) provides C5 instances with up to 25 Gbps of network bandwidth and up to 19 Gbps of dedicated bandwidth to Amazon EBS.
- Powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor

Model	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)***	EBS Bandwidth (Mbps)
c5.large	2	4	EBS-Only	Up to 10	Up to 4,750
c5.xlarge	4	8	EBS-Only	Up to 10	Up to 4,750
c5.2xlarge	8	16	EBS-Only	Up to 10	Up to 4,750

02

[Find a similar instance type](#) in AMD EPYC™.

C6g C6gn C6i C5 C5a C5n C4

C5a instances offer leading x86 price-performance for a broad set of compute-intensive workloads.

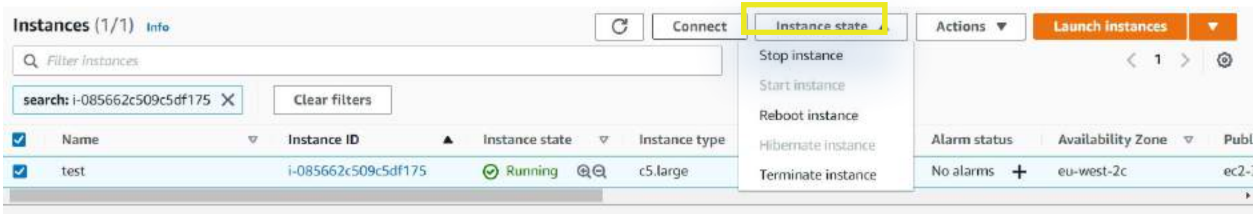
Features:

- 2nd generation AMD EPYC 7002 series processors running at frequencies up to 3.3 GHz
- Elastic Network Adapter (ENA) provides C5a instances with up to 20 Gbps of network bandwidth and up to 9.5 Gbps of dedicated bandwidth to Amazon EBS
- Powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor
- With C5ad instances, local NVMe-based SSDs are physically connected to the host server and provide block-level storage that is coupled to the lifetime of the C5a instance

Model	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)***	EBS Bandwidth (Mbps)
c5a.large	2	4	EBS-Only	Up to 10	Up to 3,170
c5a.xlarge	4	8	EBS-Only	Up to 10	Up to 3,170
c5a.2xlarge	8	16	EBS-Only	Up to 10	Up to 3,170
c5a.4xlarge	16	32	EBS-Only	Up to 10	Up to 3,170

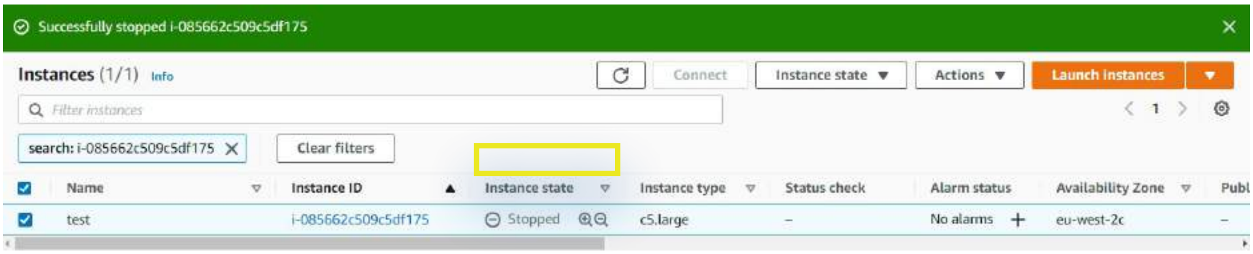
03

Stop running the existing instance.



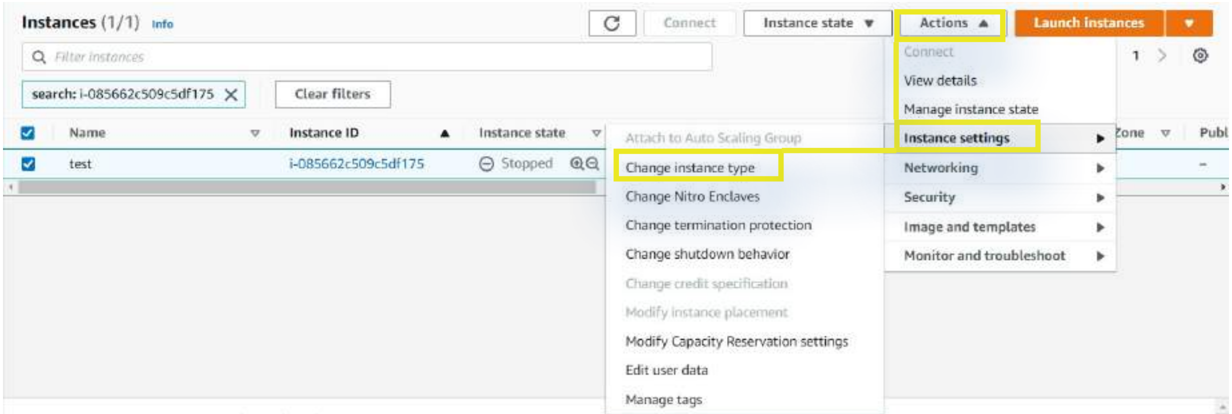
04

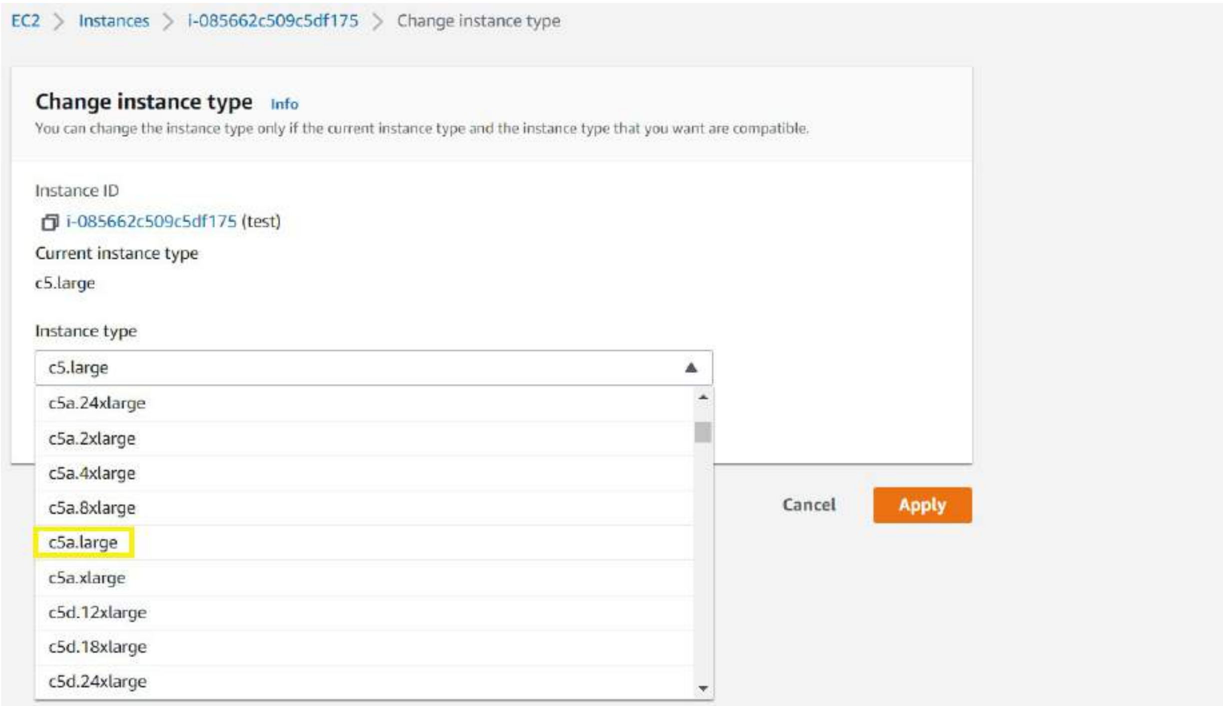
Confirm the instance has stopped.



05

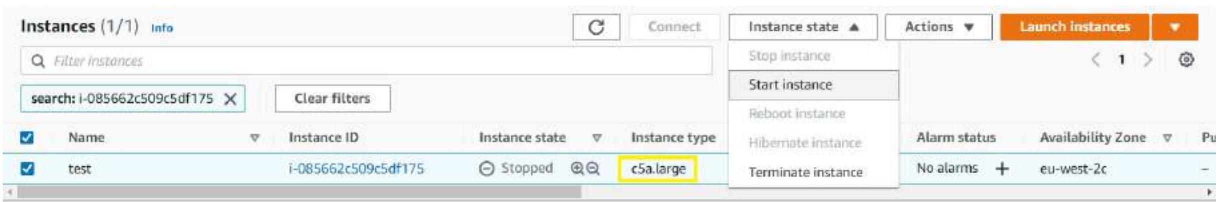
Change the instance type.





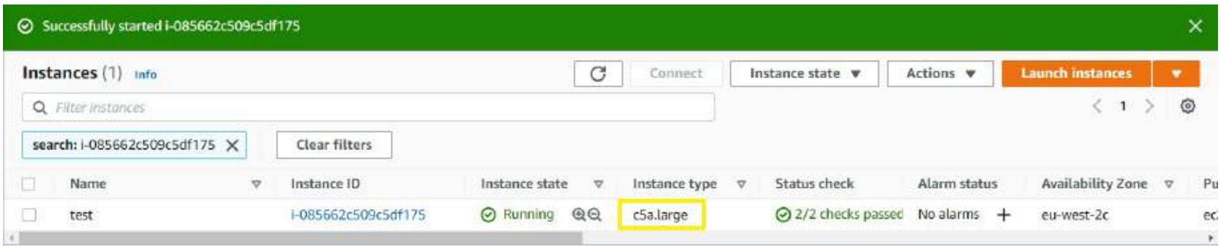
06

Start the instance.



07

Confirm the instance has started successfully.



Update from AWS CLI

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

Please follow the link to configure AWS CLI credentials to execute following commands.

The Process

01

Verify the current instance type.

```
$ aws ec2 describe-instances | jq -r '.Reservations[].Instances[] |
select(.Tags[].Key=="Name" and .Tags[].Value=="test") | .InstanceType'

c5.large
```

02

Log in to the instance and fetch the CPU information.

```
login to the instance

 ssh -i "test-user.pem" ec2-user@ec2-18-168-62-86.eu-
est-2.compute.amazonaws.com
```

```
#getting the cpu information

$ cat /proc/cpuinfo | more
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 85
model name      : Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz
stepping        : 4
microcode       : 0x1000157
cpu MHz         : 3399.896
cache size      : 25344 KB
physical id     : 0
```

03

Find a similar instance type in AMD EPYC™.

C6g C6gn C6i C5 C5a C5n C4

C5a instances offer leading x86 price-performance for a broad set of compute-intensive workloads.

Features:

- 2nd generation AMD EPYC 7002 series processors running at frequencies up to 3.3 GHz
- Elastic Network Adapter (ENA) provides C5a instances with up to 20 Gbps of network bandwidth and up to 9.5 Gbps of dedicated bandwidth to Amazon EBS
- Powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor
- With C5ad instances, local NVMe-based SSDs are physically connected to the host server and provide block-level storage that is coupled to the lifetime of the C5a instance

Model	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)***	EBS Bandwidth (Mbps)
c5a.large	2	4	EBS-Only	Up to 10	Up to 3,170
c5a.xlarge	4	8	EBS-Only	Up to 10	Up to 3,170
c5a.2xlarge	8	16	EBS-Only	Up to 10	Up to 3,170
c5a.4xlarge	16	32	EBS-Only	Up to 10	Up to 3,170

04

Stop running the instance.

```
$ aws ec2 describe-instances | jq -r '.Reservations[].Instances[] |
select(.Tags[].Key=="Name" and .Tags[].Value=="test") | .InstanceId'

i-061dd9d896587fea9

$ aws ec2 stop-instances --instance-ids i-061dd9d896587fea9

{
  "StoppingInstances": [
    {
      "InstanceId": "i-061dd9d896587fea9",
      "CurrentState": {
        "Code": 64,
        "Name": "stopping"
      },
      "PreviousState": {
        "Code": 16,
        "Name": "running"
      }
    }
  ]
}
```

05

Confirm the instance has stopped.

```
$ aws ec2 describe-instances | jq -r '.Reservations[].Instances[] |
select(.Tags[].Key=="Name" and .Tags[].Value=="test") | .State'

{
  "Code": 80,
  "Name": "stopped"
}
```

06

Log in to the instance and fetch the CPU information.

```
$ aws ec2 modify-instance-attribute --instance-id i-061dd9d896587fea9 --
instance-type "{\"Value\": \"c5a.large\"}"

$ aws ec2 describe-instances | jq -r '.Reservations[].Instances[] |
select(.Tags[].Key=="Name" and .Tags[].Value=="Test") | .InstanceType'

c5a.large
```

07

Start the new instance type.

```
$ aws ec2 start-instances --instance-ids i-061dd9d896587fea9

{
  "StartingInstances": [
    {
      "CurrentState": {
        "Code": 0,
        "Name": "pending"
      },
      "InstanceId": "i-061dd9d896587fea9",
      "PreviousState": {
        "Code": 80,
        "Name": "stopped"
      }
    }
  ]
}
```

08

Confirm the instance has started successfully.

```
$ aws ec2 describe-instances | jq -r
'.Reservations[].Instances[] | select(.Tags[].Key=="Name"
and .Tags[].Value=="test") | .State'

{
  "Code": 16,
  "Name": "running"
}
```

09

Log in to the instance and fetch the CPU information.

```
$ aws ec2 describe-instances | jq -r
'.Reservations[].Instances[] | select(.Tags[].Key=="Name"
and .Tags[].Value=="test") | .State'

{
  "Code": 16,
  "Name": "running"
}

#login to the instance

$ ssh -i "test-user.pem" ec2-user@ec2-18-168-62-86.eu-
west-2.compute.amazonaws.com
```

```
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 23
model           : 49
model name      : AMD EPYC 7R32
stepping        : 0
microcode       : 0x8301034
cpu MHz         : 2261.794
cache size      : 512 KB
physical id     : 0
```

06

MOVEMENT OF AN EC2 INSTANCE APPLICATION

HashiCorp Vault

This section demonstrates how an application running on several EC2 instances can be moved using either AWS CloudFormation or Terraform. The exercise uses the HashiCorp Vault application, an open-source tool for securely storing secrets and sensitive data in dynamic cloud environments.

The infrastructure is currently deployed in AWS EC2 instance; the underlying nodes are using AWS Intel instances. The transition process will update this infrastructure from Intel EC2 instances to AMD EPYC™ EC2 instances.

Tools

HashiCorp Vault | Stores and secures access for sensitive data using a UI, CLI or HTTP API.

HashiCorp Terraform | Open-source IaC software that provides a consistent CLI workflow to manage cloud services.

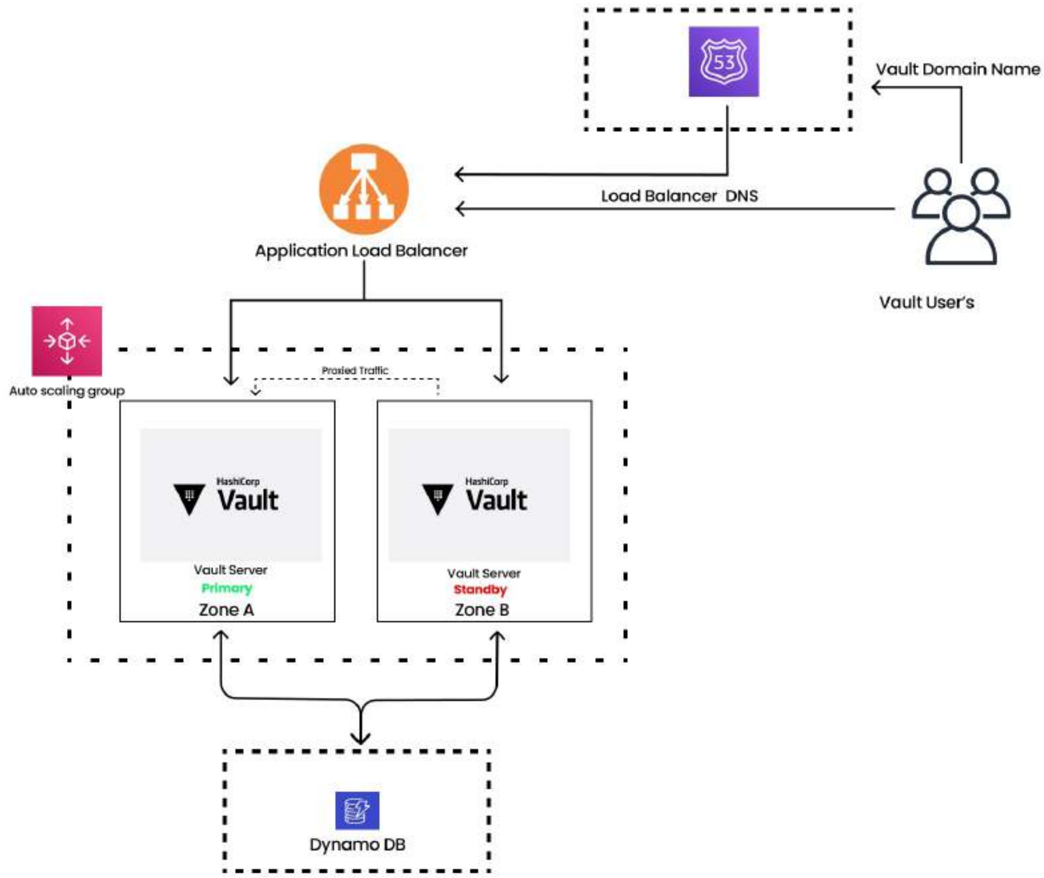
AWS EC2 | Provides a wide selection of instance types optimised to fit different use cases.

AWS DynamoDB | Fully managed, serverless, key-value NoSQL database for high-performance applications.

The Vault Servers, hosted on EC2 instances, are created using an Auto Scaling Group (ASG). So, if any instance goes down, it's automatically replaced. It also uses DynamoDB in the on-demand mode which means that no management is required for server capacity, storage, or throughput.

When Vault is in High Availability Mode, it enables multiple Vault Servers in the deployment. At any given time, one Vault Server will be "active" and serving the incoming requests/writing/reading the encrypted data. Other Vault Servers are put in "stand-by" mode in case the "active" server fails.

Application Architecture in AWS

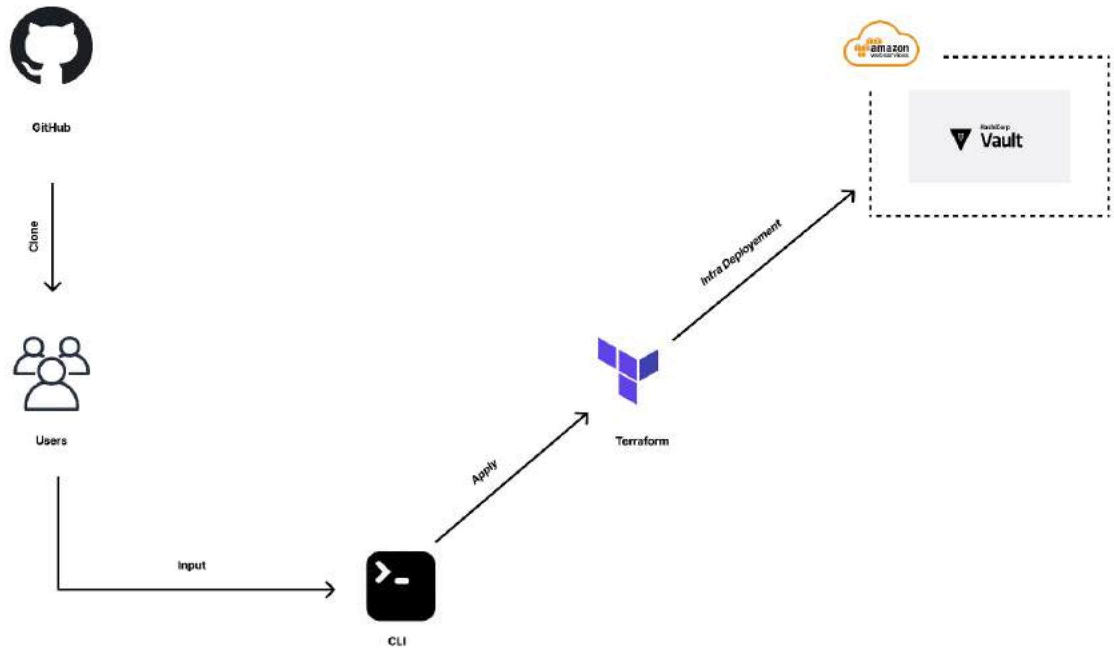


Deployment with Terraform

Terraform is an open-source, infrastructure-as-code (IaC) software tool that allows you to build, change and version infrastructure, both safely and efficiently. It can be used to manage existing service providers as well as custom in-house solutions, for both low-level (eg compute instances, storage, networking) and high-level components (eg DNS entries, SaaS features).

This repository contains a set of Terraform files for deploying a Vault cluster on AWS. HashiCorp Vault helps to store the infrastructural secrets and credentials in a highly available setup.

Deployment Architecture



Clone the application GitHub repository, and follow the Readme.md file to configure and deploy the HashiCorp Vault in AWS with the help of Terraform.

This project requires that you have Terraform 0.14+ installed. Both deployment and management should be done through Terraform.

The deployment process is done using a Terraform template, by cloning the repository into CLI and modifying the required parameters, then executing a command, the entire infrastructure will be provisioned. Users can access the vault dashboard via Route53 or can use load balancer DNS.

Terraform will create a base infrastructure containing:

- EC2 instances
- DynamoDB
- S3 bucket
- KMS Key
- an Application Load Balancer
- an Autoscaling Group
- Route53 subdomain entry
- VPC and its components

Terraform also executes a user-data which helps to initialise the vault and allows it to auto-unseal with the help of a KMS key. The root token and key shards will be uploaded to an S3-bucket that has to be created by Terraform.

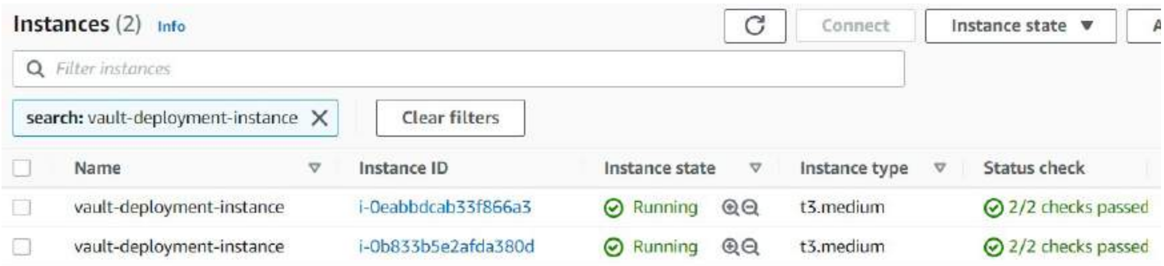
The DynamoDB storage will provide high availability and is used to persist Vault's data in the DynamoDB table.

The Process

Moving from t3.medium instance infrastructure to an AMD-based t3a.medium instance.

01

AWS EC2 Instance View: Instances are currently running on Intel t3.medium.



02

Clone the repository.

```
$ git clone https://github.com/stackgenie/stackgenie-devops-amd01.git &&
cd stackgenie-devops-amd01
```

03

Go to the Terraform variable file, update the file (variables.tf) with the "Instance type" to AMD EPYC™ (t3a.medium).

```
variable "vault_instance_type" {
  description = "The EC2 instance size of the vault."
  type = string
  default = "t3a.medium"
}
```

04

Once the file is updated, execute the below commands.

```
$ terraform init

$ terraform plan
```

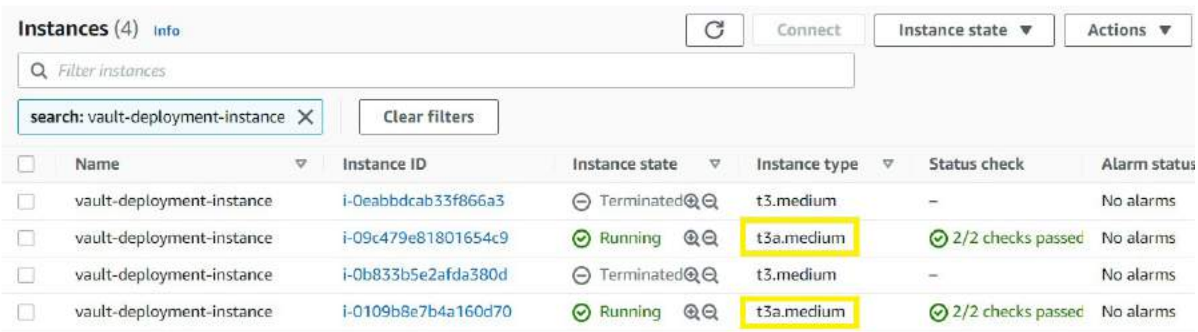
Terraform Plan Output:

```
# aws_launch_template.vault_instance will be updated in-place
~ resource "aws_launch_template" "vault_instance" {
      id                    = "lt-08add968a20a81c32"
    ~ instance_type         = "t3.medium" -> "t3a.medium"
    ~ latest_version        = 1 -> (known after apply)
```

```
$ terraform apply
```

05

AWS Instance View: Instance type has been modified from Intel powered t3.medium to AMD EPYC™ powered t3a.medium.


Instances (4) Info | Connect | Instance state ▼ | Actions ▼

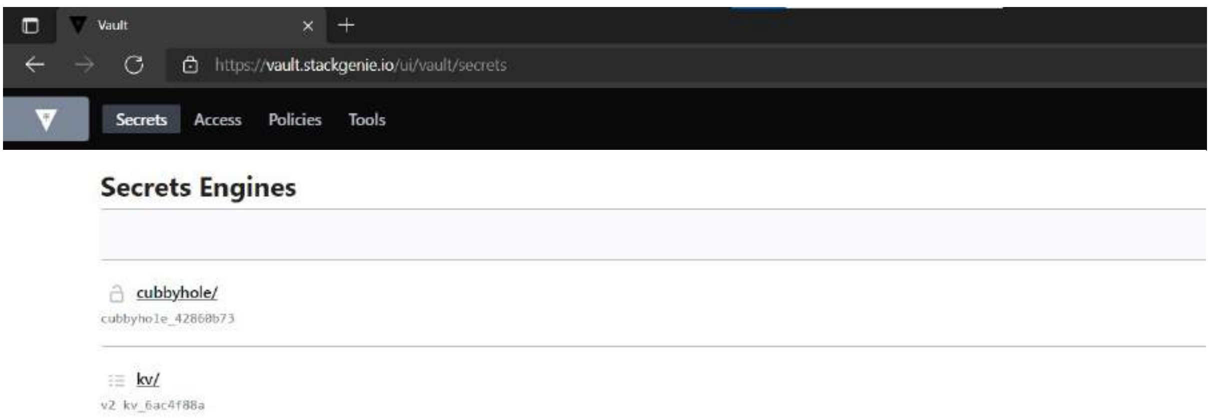
Filter instances

search: vault-deployment-instance | Clear filters

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
vault-deployment-instance	i-0eabbdcab33f866a3	Terminated	t3.medium	–	No alarms
vault-deployment-instance	i-09c479e81801654c9	Running	t3a.medium	2/2 checks passed	No alarms
vault-deployment-instance	i-0b833b5e2afda380d	Terminated	t3.medium	–	No alarms
vault-deployment-instance	i-0109b8e7b4a160d70	Running	t3a.medium	2/2 checks passed	No alarms

06

Vault Browser View: Verify the application is working as expected.



07

The result is a successful transition of EC2 instances from Intel-based processors to AMD variants using Terraform.

Deployment with CloudFormation, Packer and Ansible

CloudFormation is an AWS managed service that allows you to manage the infrastructure in AWS using templates. As the name suggests, it is an Infrastructure as code (IaC) tool. CloudFormation is used for automating the deployment and configuration of the majority of services in AWS.

Packer is an open source tool from HashiCorp that can be used to create golden images from a single source of configuration.

Ansible is a highly versatile open source tool. It can handle configuration management, application deployment, cloud provisioning, ad-hoc task execution, network automation and multi-node orchestration.

Tools

HashiCorp Vault | Stores and secures access for sensitive data using a UI, CLI or HTTP API.

HashiCorp Packer | Creates identical machine images for multiple platforms from a single source configuration.

AWS CloudFormation | Treats infrastructure as code to model, provision and manage AWS and third-party resources.

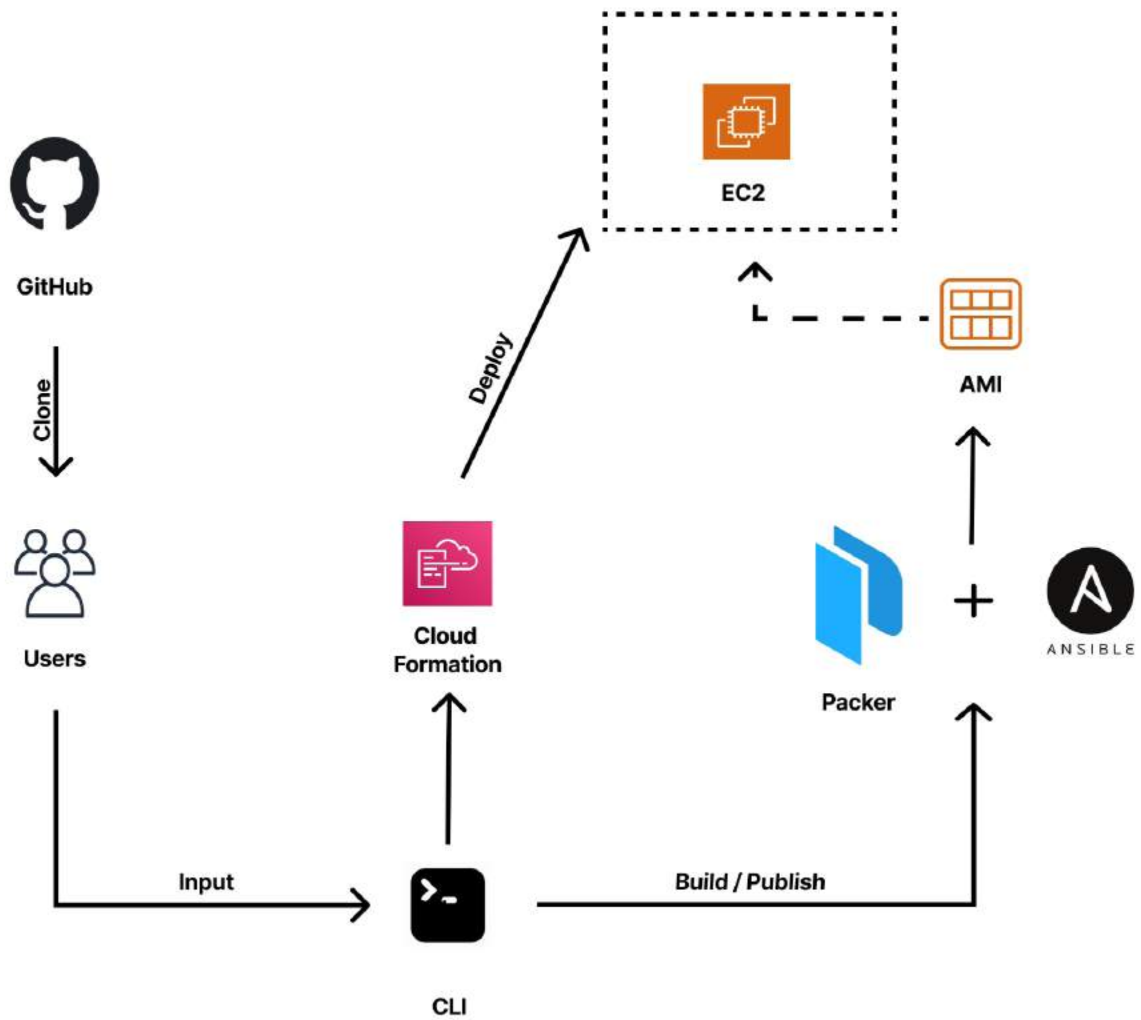
AWS EC2 | Provides a wide selection of instance types optimised to fit different use cases.

AWS DynamoDB | Fully managed, serverless, key-value NoSQL database for high-performance applications.

CloudFormation Templates

The CloudFormation templates (vault_cfn.yaml) available in the GitHub repository will deploy the application in AWS. This CloudFormation template deploys a VPC with both public and private subnet across two Availability Zones. It also provisions an instance backed by an autoscaling group which is using a custom Amazon Machine Image (AMI) created with Packer. Finally, the CloudFormation template will create an application load balancer, auto scaling group, DynamoDB table, SSM Parameter Store, KMS key and Route53 subdomain entry.

Deployment Architecture



Clone the application GitHub repository, and follow the Readme.md file to deploy the application. The packer builder command will provision an AMI with the help of the Ansible provisioner.

Use a custom build AMI, as this application is deployed into an EC2 instance, by using HCP Packer with Ansible provisioner. The deployment process can start once the custom AMI is ready. This is done using a CloudFormation template, by cloning the repository into CLI and modifying the required parameters, then executing a command, the entire infrastructure will be provisioned. Users can access the vault dashboard via Route53 or can use load balancer DNS.

The user-data in the Launch Template will initialise the Vault cluster and upload the root keys and recovery keys in the SSM parameter store. An encryption key from AWS Key Management Services (KMS) will help to auto-unseal Vault.

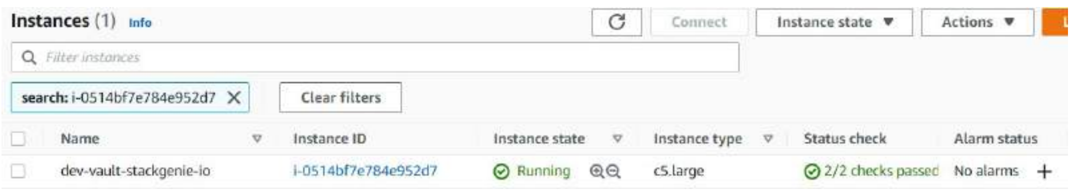
Whilst, the DynamoDB storage backend supports high availability and is used to persist Vault's data in the DynamoDB table.

The Process

In this example, the application is running on an Intel-powered c5.large EC2 instance. To move the application to an AMD EPYC™ powered c5a.large instance, update the "Instance Type" parameter on the CloudFormation stack; the update will redeploy the application and change the EC2 instance type.

01

AWS EC2 Instance View: Current infrastructure is running on Intel c5.large instance.

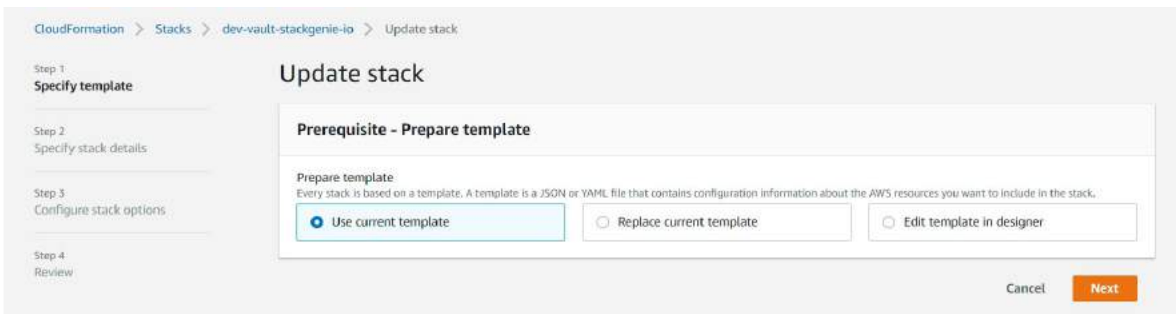


02

AWS CFN Stack View: To update the current CloudFormation Stack that is deployed, click on 'update'.

03

AWS CFN Stack Actions: Click on 'use current template'.

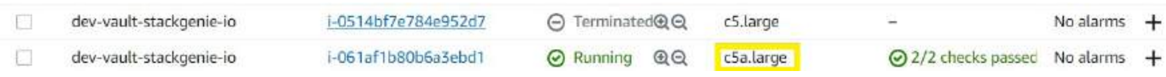


04

AWS CFN Stack actions: Change the instance type to AMD EPYC™ powered c5a.large. Run the Stack by clicking 'next', then 'finish'.

05

AWS EC2 Instance View: The Instance Type has been modified from Intel-powered c5.large to AMD EPYC™ powered c5a.large.



06

Vault Browser View: Verify the application is working as expected.

07

The result is a successful transition of EC2 instances from Intel-based processors to AMD variants using CloudFormation.

07

MOVEMENT OF A MICROSERVICE WEB APPLICATION WITH DATABASES

e-Commerce Application

In this example, to aid the demonstrations and testing of microservices and cloud-native technologies, the application for the purpose of migration is an e-commerce application. As an online shop that sells products, it is a multi-tiered application that has both a web front-end, that's user-facing, and a database back-end. Please refer to the online shop application GitHub repository.

The application is built using Spring Boot, Go kit and Node.js and is packaged in Docker containers. You can read more about the application design.

Tools

HashiCorp Terraform | Open-source IaC software that provides a consistent CLI workflow to manage cloud services.

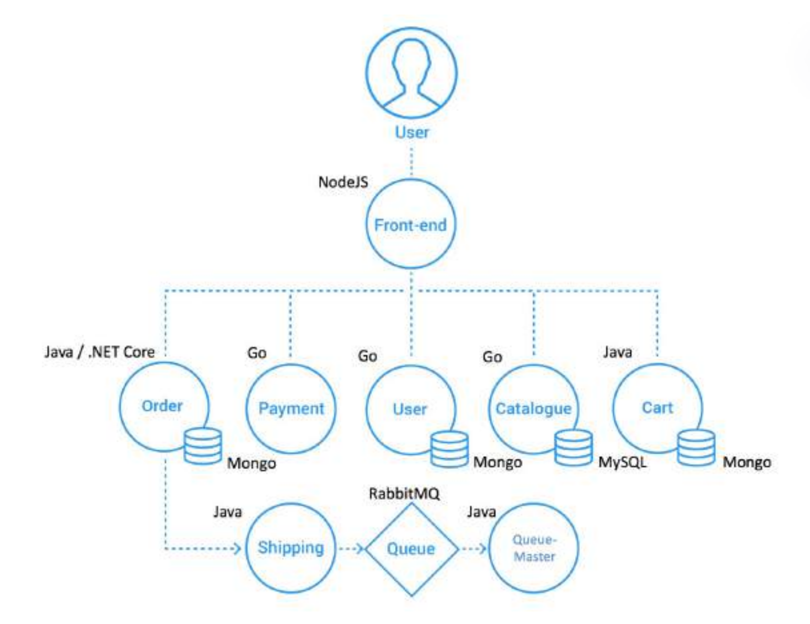
Amazon EKS | Open-source system for automating deployment, scaling and management of containerised applications.

Argo CD | A declarative, GitOps continuous delivery tool for Kubernetes.

NGINX ingress controller | Ingress exposes HTT and HTTPS routes from outside the cluster to services with the cluster. An ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

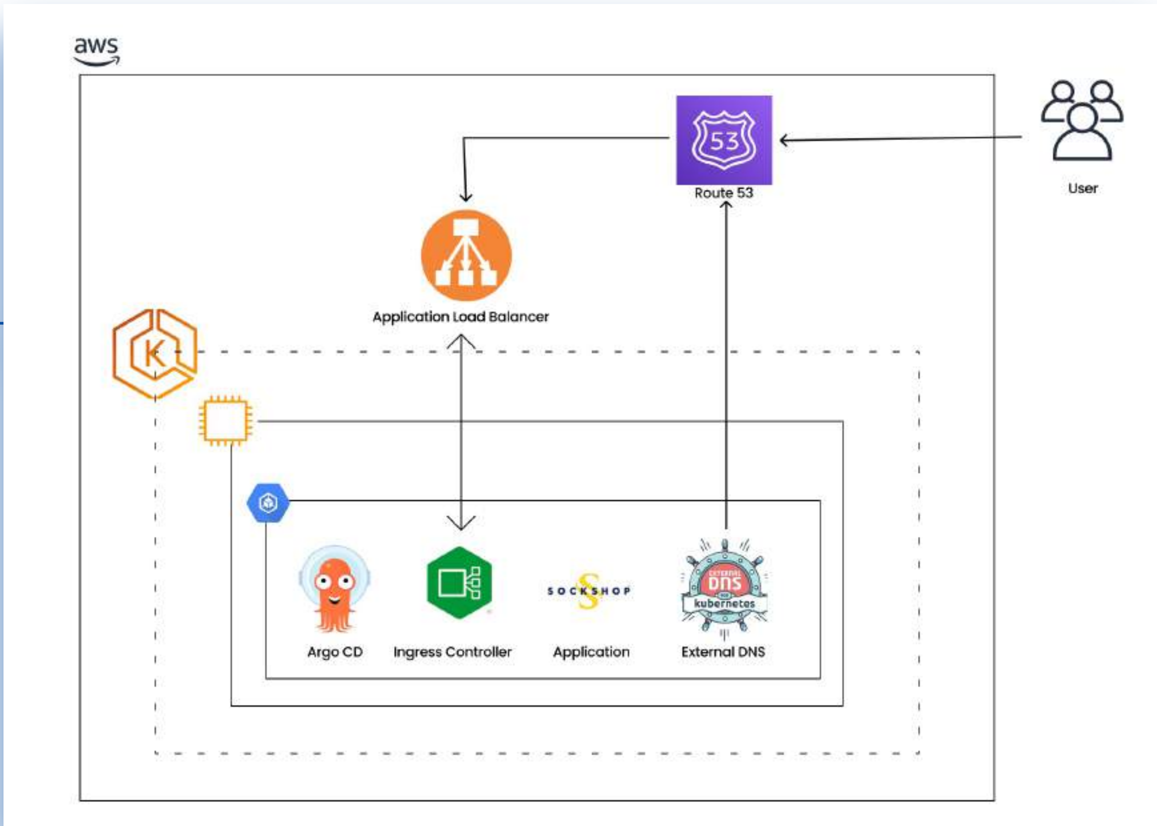
ExternalDNS | Synchronises exposed Kubernetes Services and Ingresses with DNS providers, and makes Kubernetes resources discoverable via public DNS servers.

e-Commerce Application Design



The architecture of the demo microservices application was intentionally designed to provide as many microservices as possible, as well as being polyglot to exercise a number of different technologies. The microservices are roughly defined by the function in an e-Commerce site. All services communicate using REST over HTTP. This was chosen due to the simplicity of development and testing.

The Application containers are deployed on EKS Cluster with CI/CD tool Argo CD so the application deployment and lifecycle management should be automated, auditable, and easy to understand. It also uses some supporting tools NGINX ingress controller, and external DNS provisioner. All these microservices are scheduled on Amazon EKS managed node groups to automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters and all managed nodes are provisioned as part of an Amazon EC2 Auto Scaling group that's managed for you by Amazon EKS.



Application Structure in EKS

```
$ kubectl get all -n sock-shop

NAME                                 READY   STATUS    RESTARTS   AGE
pod/carts-b4d4ffb5c-n4kq9            1/1     Running   0          17m
pod/carts-b4d4ffb5c-nfgll            1/1     Running   0          61m
pod/carts-db-6c6c68b747-f7g6l        1/1     Running   0          61m
pod/carts-db-6c6c68b747-fsjwz        1/1     Running   0          17m
pod/catalogue-759cc6b86-9g694        1/1     Running   0          61m
pod/catalogue-759cc6b86-rtxrg        1/1     Running   0          17m
pod/catalogue-db-96f6f6b4c-567gn     1/1     Running   0          61m
pod/catalogue-db-96f6f6b4c-72plj     1/1     Running   0          17m
pod/front-end-5c89db9f57-n7s9w       1/1     Running   0          61m
pod/front-end-5c89db9f57-nwnb9       1/1     Running   0          17m
pod/orders-7664c64d75-2vkfk          1/1     Running   0          17m
pod/orders-7664c64d75-kmtqd          1/1     Running   0          61m
pod/orders-db-659949975f-7td6n       1/1     Running   0          61m
pod/orders-db-659949975f-tfzrp       1/1     Running   0          17m
pod/payment-7bcdbf45c9-9mkvd         1/1     Running   0          17m
pod/payment-7bcdbf45c9-q69xd         1/1     Running   0          61m
pod/queue-master-5f6d6d4796-8j2z8    1/1     Running   0          17m
pod/queue-master-5f6d6d4796-xgjd6    1/1     Running   0          61m
pod/rabbitmq-5bcbb547d7-c6jgt        2/2     Running   0          61m
pod/rabbitmq-5bcbb547d7-fqspr        2/2     Running   0          17m
pod/session-db-7cf97f8d4f-d5xpw      1/1     Running   0          61m
pod/session-db-7cf97f8d4f-gb8kq      1/1     Running   0          17m
pod/shipping-7f7999ffb7-vj5fl        1/1     Running   0          61m
pod/shipping-7f7999ffb7-x4hlh        1/1     Running   0          17m
pod/user-68df64db9c-qkmvj            1/1     Running   0          61m
pod/user-68df64db9c-xclps            1/1     Running   0          17m
pod/user-db-6df7444fc-pm9bs          1/1     Running   0          61m
pod/user-db-6df7444fc-r2qk6          1/1     Running   0          17m

NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
service/carts          ClusterIP   172.20.192.246   <none>        80/TCP              61m
service/carts-db       ClusterIP   172.20.101.125   <none>        27017/TCP           61m
service/catalogue      ClusterIP   172.20.122.18    <none>        80/TCP              61m
service/catalogue-db   ClusterIP   172.20.103.30    <none>        3306/TCP            61m
service/front-end      NodePort    172.20.91.62     <none>        80:30001/TCP        61m
service/orders         ClusterIP   172.20.248.186   <none>        80/TCP              61m
service/orders-db      ClusterIP   172.20.248.59    <none>        27017/TCP           61m
service/payment        ClusterIP   172.20.186.179   <none>        80/TCP              61m
service/queue-master   ClusterIP   172.20.173.120   <none>        80/TCP              61m
service/rabbitmq       ClusterIP   172.20.140.122   <none>        5672/TCP,9090/TCP   61m
service/session-db     ClusterIP   172.20.102.127   <none>        6379/TCP            61m
service/shipping       ClusterIP   172.20.199.86    <none>        80/TCP              61m
service/user           ClusterIP   172.20.47.126    <none>        80/TCP              61m
service/user-db        ClusterIP   172.20.0.18      <none>        27017/TCP           61m

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/carts           2/2     2            2           61m
deployment.apps/carts-db        2/2     2            2           61m
deployment.apps/catalogue       2/2     2            2           61m
deployment.apps/catalogue-db    2/2     2            2           61m
deployment.apps/front-end       2/2     2            2           61m
deployment.apps/orders          2/2     2            2           61m
deployment.apps/orders-db       2/2     2            2           61m
deployment.apps/payment         2/2     2            2           61m
deployment.apps/queue-master    2/2     2            2           61m
deployment.apps/rabbitmq        2/2     2            2           61m
deployment.apps/session-db      2/2     2            2           61m
deployment.apps/shipping        2/2     2            2           61m
deployment.apps/user            2/2     2            2           61m
deployment.apps/user-db         2/2     2            2           61m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/carts-b4d4ffb5c            2         2         2       61m
replicaset.apps/carts-db-6c6c68b747        2         2         2       61m
replicaset.apps/catalogue-759cc6b86        2         2         2       61m
replicaset.apps/catalogue-db-96f6f6b4c     2         2         2       61m
replicaset.apps/front-end-5c89db9f57       2         2         2       61m
replicaset.apps/orders-7664c64d75          2         2         2       61m
replicaset.apps/orders-db-659949975f       2         2         2       61m
replicaset.apps/payment-7bcdbf45c9         2         2         2       61m
replicaset.apps/queue-master-5f6d6d4796    2         2         2       61m
replicaset.apps/rabbitmq-5bcbb547d7        2         2         2       61m
replicaset.apps/session-db-7cf97f8d4f      2         2         2       61m
replicaset.apps/shipping-7f7999ffb7        2         2         2       61m
replicaset.apps/user-68df64db9c            2         2         2       61m
replicaset.apps/user-db-6df7444fc          2         2         2       61m
```

The application is currently deployed in EKS, with the underlying EC2 instances using AWS Intel c5.large instances.

The transition process will update this infrastructure from c5.large instances to c5a.large instances that are based on AMD EPYC™.

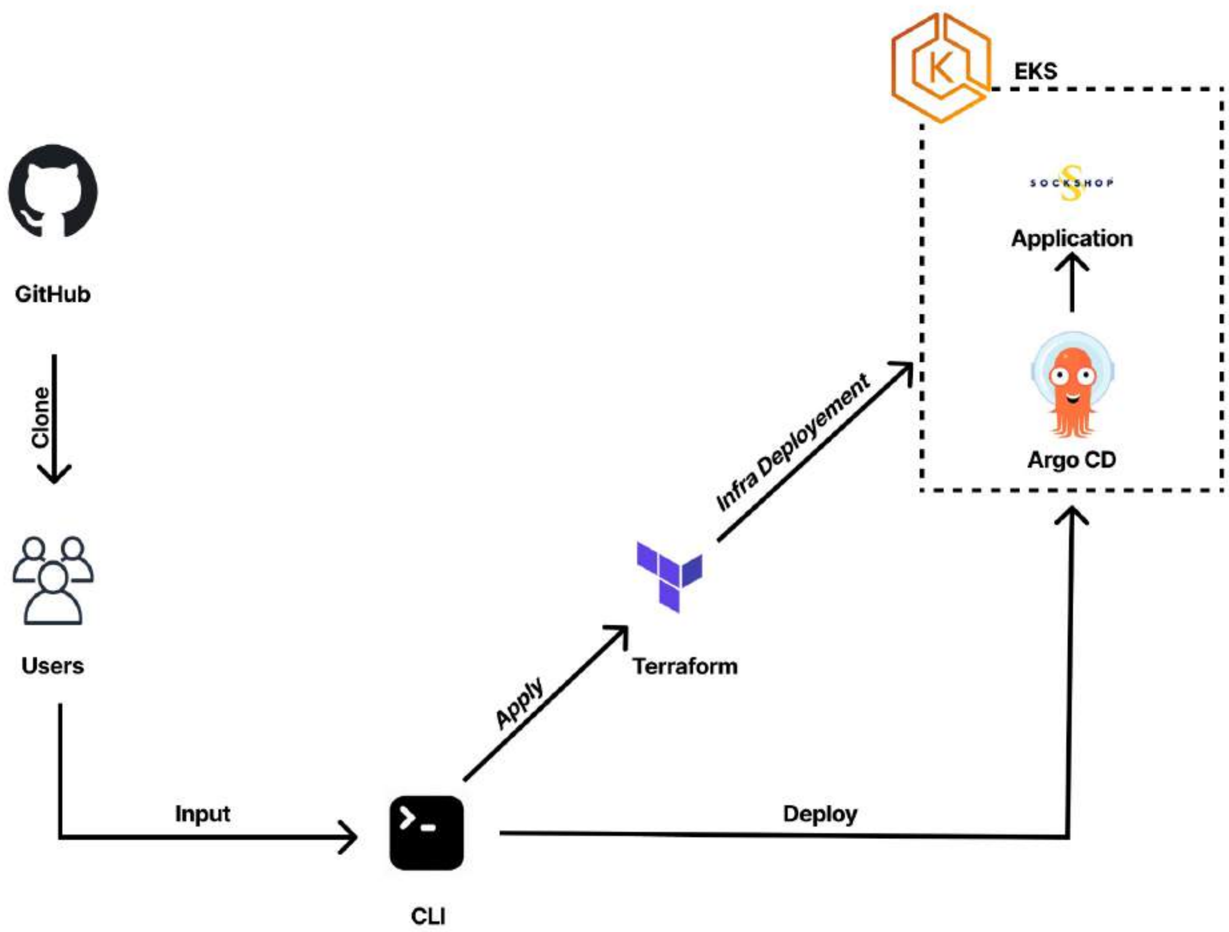
Deployment with Terraform

In this example, Terraform is used only for deploying the AWS infrastructure. The AWS infrastructure includes EKS cluster, VPCs and its components.

Also, Argo CD is already deployed on the cluster using an Argo manifest. The microservices application is deployed with Argo CD. Optional services like "Nginx ingress controller along with ExternalDNS" will also be deployed on the cluster.

The movement of these applications on EKS node group from AWS Intel c5.large instance to AWS AMD EPYC™ powered c5a.large can be achieved by changing the instance type on the Terraform variable file.

Deployment Architecture



Clone the application repository and follow the readme to deploy the application in EKS. This application is deployed using tools such as HashiCorp Terraform, AWS EKS (Elastic Kubernetes Services), and Argo CD.

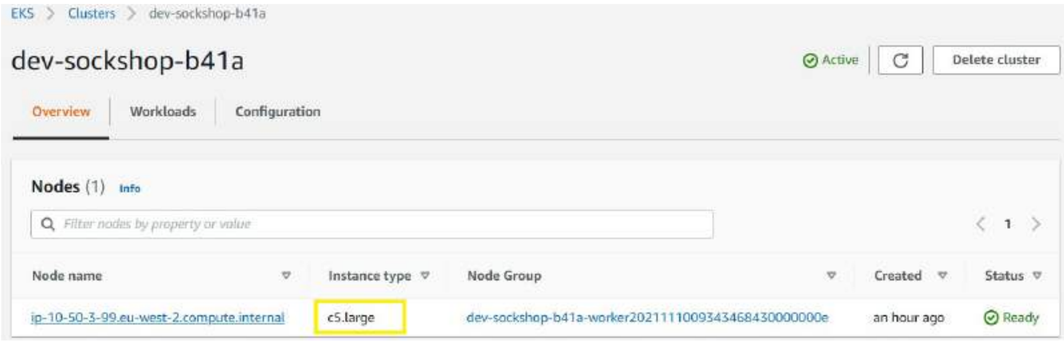
This project requires Terraform 0.14+ and Argo CD CL. The deployment process uses a Terraform template, by cloning the repository into CLI and modifying the required parameters, then executing a command, the entire infrastructure will be provisioned. Once the infrastructure is ready, the application is deployed using Argo CD CLI. Users can access the application via Route53 or can port-forward the application to localhost.

The Process

Moving from c5.large instance to an AMD-based c5a.large instance.

01

AWS EC2 Instance View: EC2 instances (EKS nodes) are running on Intel c5.large.



02

Clone the repository.

```
$ git clone https://github.com/stackgenie/stackgenie-devops-amd03.git &&
cd stackgenie-devops-amd03
```

03

For changing the instance type from Intel-powered c5.large to AMD EPYC™ powered c5a.large, update the "node_instance_type" in Terraform variable file (variables.tf).

```
variable "node_instance_type" {
      Default = "c5a.large"
}
```

04

Applying Terraform changes will update the instance type to c5a.large.

```
$ terraform init

$ terraform plan
```

```
# module.eks.module.node_groups.aws_eks_node_group.workers["worker"] must
be replaced
+/- resource "aws_eks_node_group" "workers" {
    ~ instance_types         = [ # forces replacement
                      - "c5.large",
                        + "c5a.large",
```

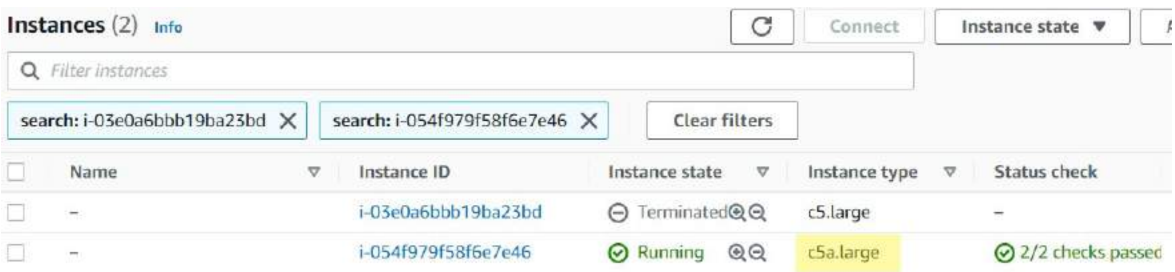
```
$ terraform apply
```

05

AWS Console Cluster View: The EC2 instances are modified from Intel c5.large to AMD EPYC™ powered c5a.large.

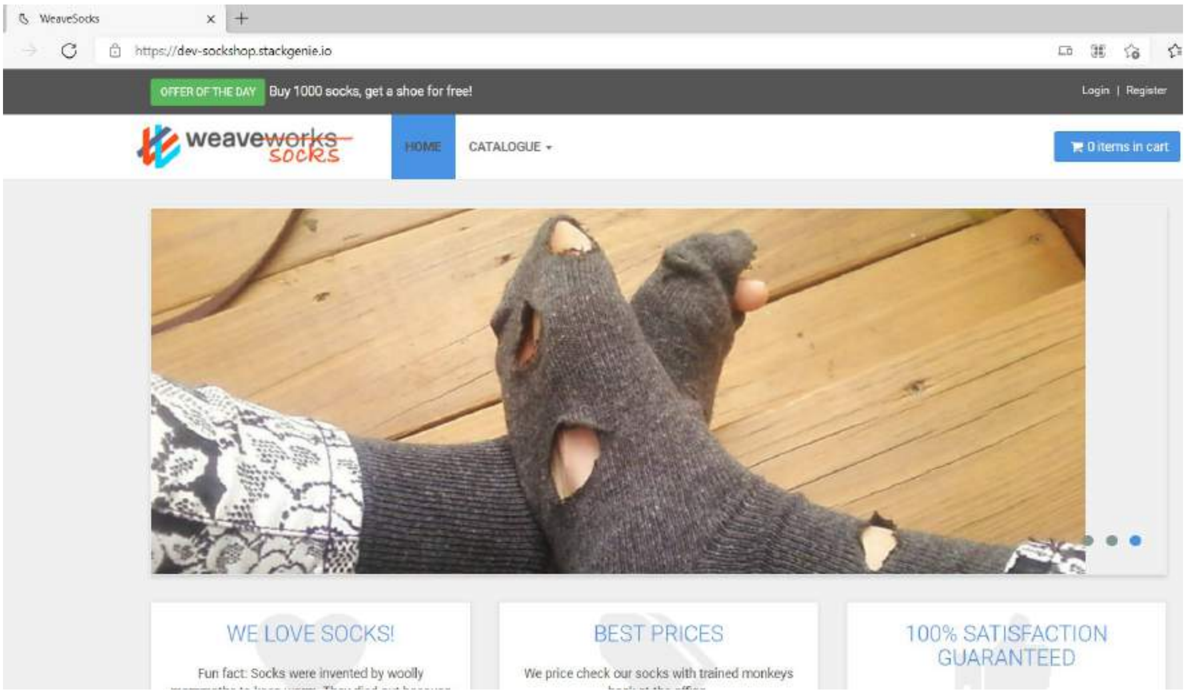
06

AWS Console instance view: The EC2 instances after the transition.



07

Browser result: Application is working as expected after the instance type is changed from c5.large to c5a.large.



Deployment with CloudFormation

As in the previous example, Argo CD is deployed on the cluster using an Argo manifest. The microservices application is deployed with Argo CD. Optional services like "Nginx ingress controller along with ExternalDNS" will also be deployed on the cluster.

For migrating the application from the AWS Intel c5.large instance to AWS AMD EPYC™ powered c5a.large, update the CloudFormation stack with a new instance type (c5a.large).

Argo CD

Argo CD automates the deployment of the desired application states in the specified target environments.

Application deployments can track updates to branches, tags, or pinned to a specific version of manifests at a Git commit.

Tools

AWS CloudFormation | Treats infrastructure as code to model, provision and manage AWS and third-party resources.

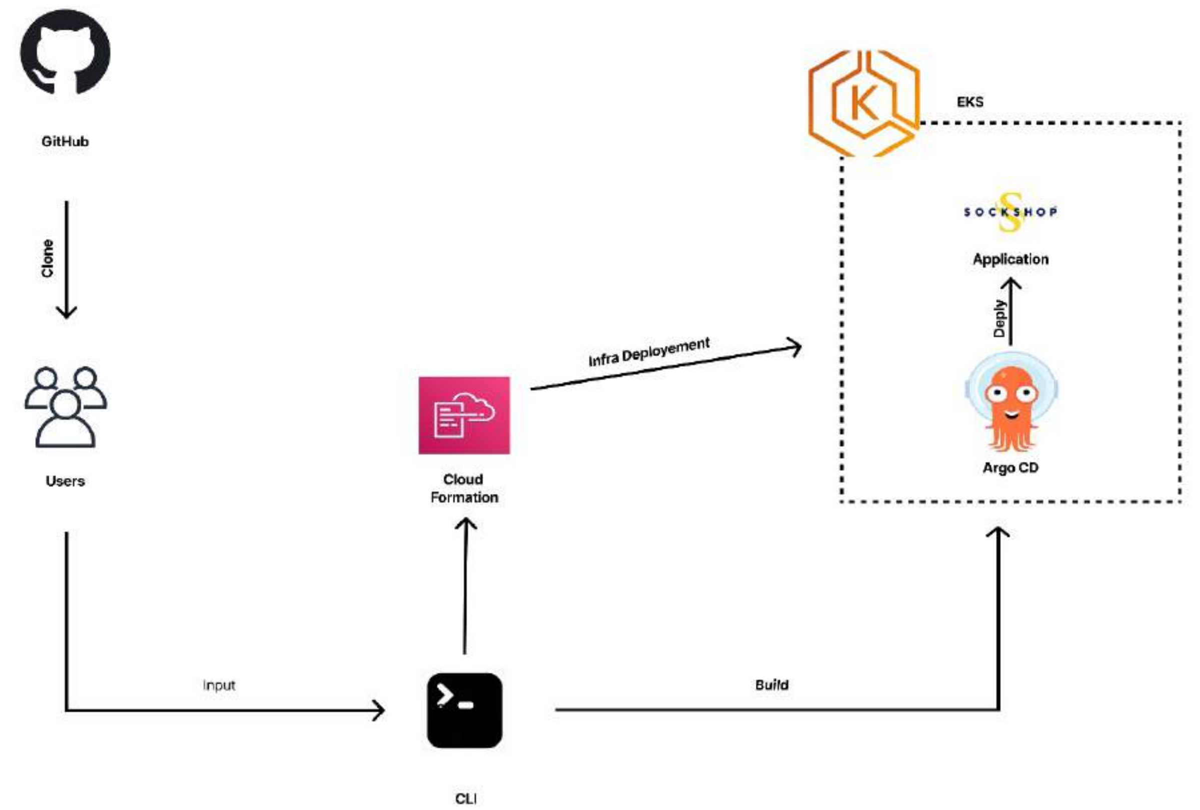
Amazon EKS | Open-source system for automating deployment, scaling and management of containerised applications.

Argo CD | A declarative, GitOps continuous delivery tool for Kubernetes.

NGINX ingress controller | Ingress exposes HTT and HTTPS routes from outside the cluster to services with the cluster. An ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

ExternalDNS | Synchronises exposed Kubernetes Services and Ingresses with DNS providers, and makes Kubernetes resources discoverable via public DNS servers.

Deployment Architecture



Clone the application repository and follow the readme to deploy the infrastructure and application. Deploying the Cloud Formation Template will create the AWS infrastructure for the application. The AWS Infrastructure contains a VPC, EKS cluster, and EKS NodeGroup.

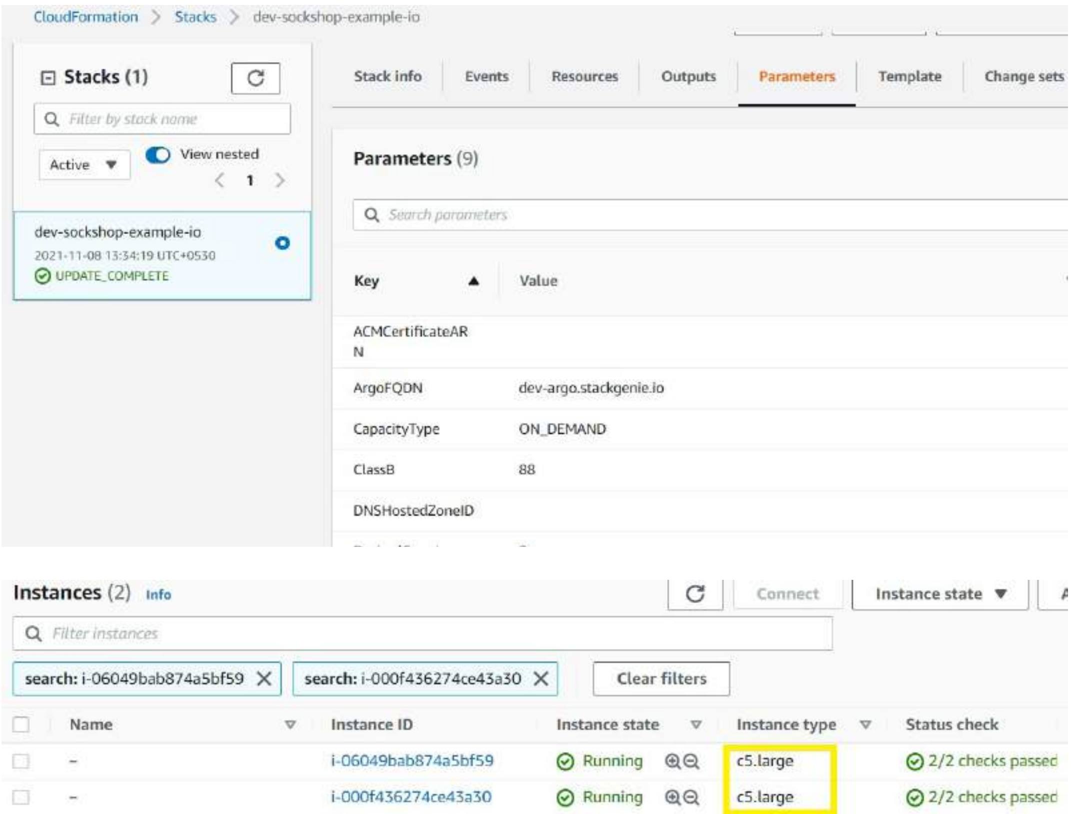
The application is deployed into EKS, here we are using CloudFormation for infrastructure deployment and Argo CD for the application deployment. First, clone the repository and modify the required parameters, then execute the command, the entire infrastructure will be provisioned. Once we have the infrastructure, next, deploy an application using Argo CD. Users can access the application via Route53 or can port-forward the application to localhost.

The Process

Moving from c5.large instance infrastructure to an AMD-based c5a.large instance.

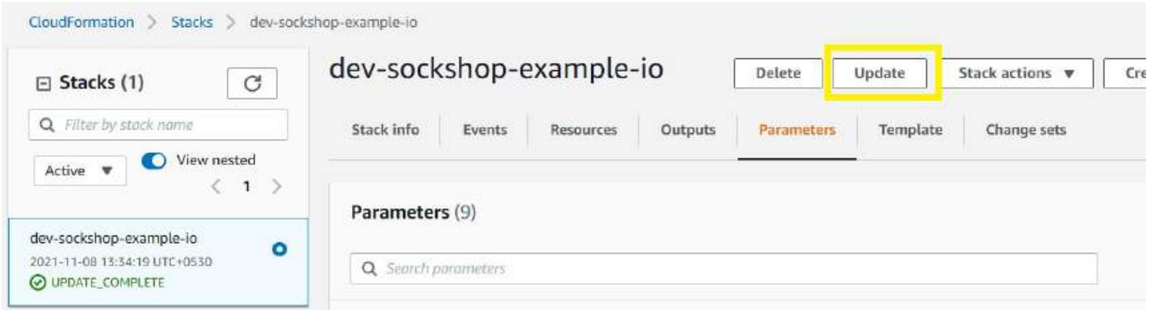
01

AWS CFN stack console view: The Intel EC2 instance is c5.large as highlighted below.



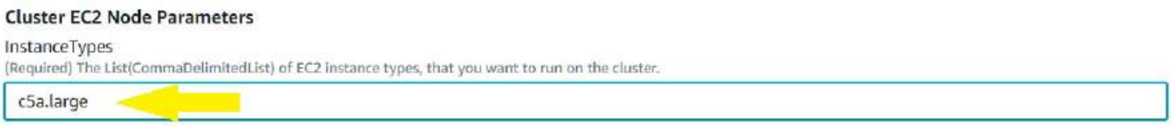
02

AWS console actions: Change the instance type to AMD EPYC™ powered c5a.large by updating the current template:



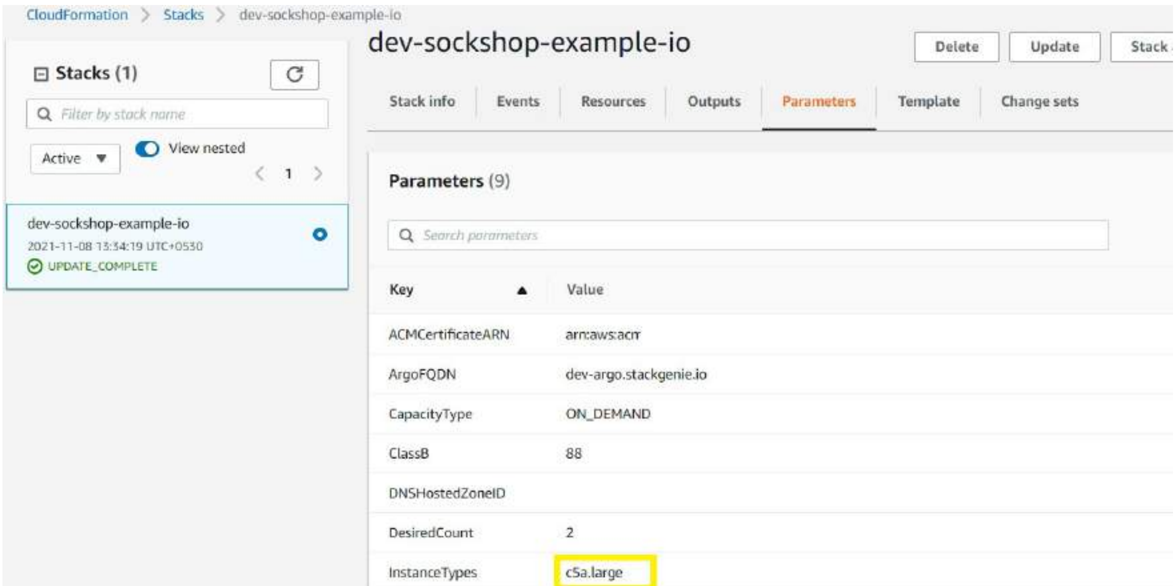
03

AWS console actions: Choose 'use current template' and change the instance type to AMD EPYC™ powered c5a.large.



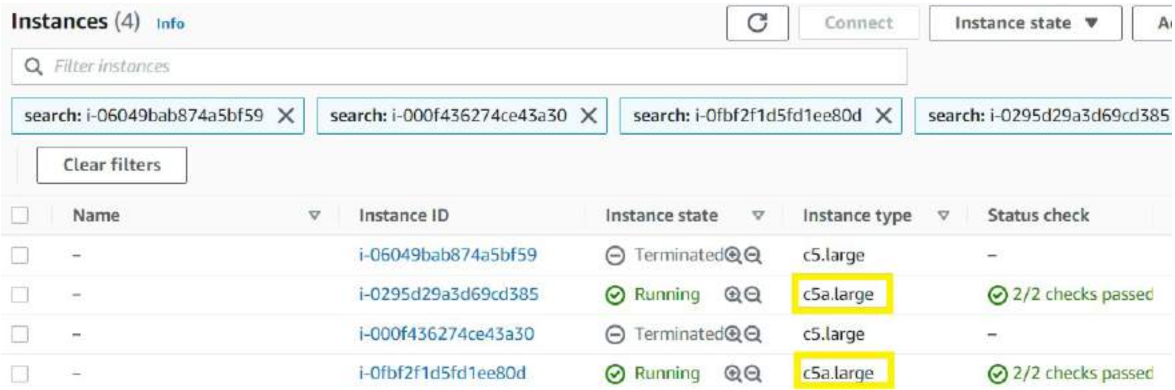
04

AWS console view: The EC2 instances are modified from Intel c5.large to AMD EPYC™ powered c5a.large.



05

AWS instance view: The EC2 instances after the transition.



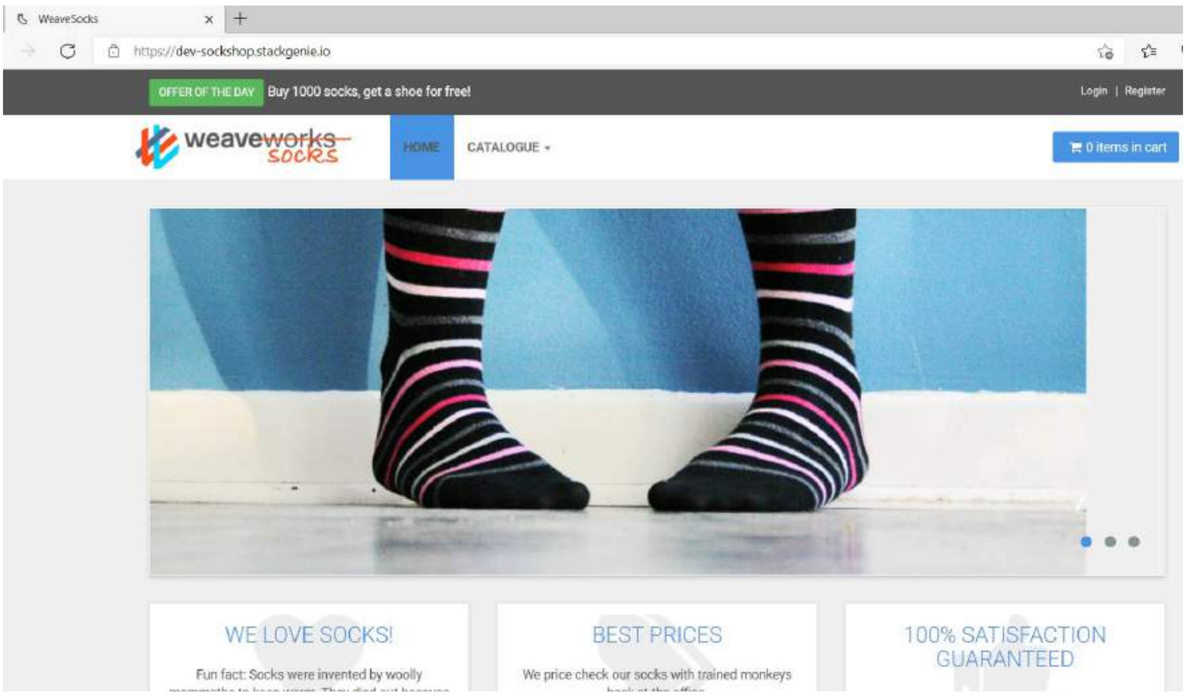
Instances (4) Info

search: i-06049bab874a5bf59 | search: i-000f436274ce43a30 | search: i-0fbf2f1d5fd1ee80d | search: i-0295d29a3d69cd385

Name	Instance ID	Instance state	Instance type	Status check
–	i-06049bab874a5bf59	Terminated	c5.large	–
–	i-0295d29a3d69cd385	Running	c5a.large	2/2 checks passed
–	i-000f436274ce43a30	Terminated	c5.large	–
–	i-0fbf2f1d5fd1ee80d	Running	c5a.large	2/2 checks passed

06

Browser result: The application is working as expected after moving the instance type from c5.large to c5a.large.



stackgenie

CONCLUSION

Whilst Intel processors have been the default choice for running instances on Amazon EC2 for well over a decade, the launch of AMD EPYC™ processors in 2018 provided Amazon customers with alternative options for running workloads.

Having more availability and choice means AWS users can optimise for performance and cost, as well as right-sizing their workloads by choosing from a wide variety of Intel and AMD-based options.

Same x86 Architecture

Both Intel and AMD EPYC™ processors use the same x86 architecture, which means, in most situations, applications running on existing EC2 instances can transition from one to the other, seamlessly.

A little change can save customers up to 10% of their compute costs.

Vinayak Kumar
Stackgenie Founder and CEO

This white paper demonstrates the ease of moving from Intel to AMD-based instances. As demonstrated through the various examples, a little change can go a long way. A single line of code can save customers up to 10% of their compute costs.

For AWS customers looking at price-optimised compute options, AMD EPYC™ provides greater flexibility when looking at right-sizing instances.

There is also further excitement with the launch of the third generation of AMD EPYC™ processors. Customers will have even more flexibility and choice with the launch of R6a, C6a and M6a instances.

WHAT WE DO?

HOW WE SUPPORT YOU

Augment and enhance your people capabilities throughout your digital transformation journey with us partnering and working as an extension of your team.

STREAMLINE

Ensure holistic transformation for optimal performance.

FINANCE

Keep digital transformation focused on budget-friendly practices.

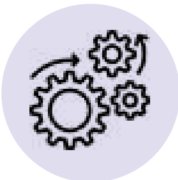
EXPERTS

Bring in expert talent to drive innovation.

ACCELERATE

Ensure holistic transformation for optimal performance.

Take advantage of our comprehensive, complementary and collaborative support matrix.



INSIGHT

Gain data-backed insights from customer preferences.

INNOVATE

Go beyond existing technologies make your business agile, secure and streamlined.

STRATEGY

Identify custom strategies to give you a competitive edge.

CUSTOM

Augment your existing business systems with tailored support plans.

stackgenie

Toll-Free: 0330-133-4519

info@stackgenie.io | hello@stackgenie.io

United Kingdom | 71-75 Shelton Street, London, United Kingdom, WC2H 9JQ

India | Orchestra Project, Laham Commercial Complex, Trivandrum, Kerala, India 695582