AWS Technical Guide

Replatform .NET Applications with Windows Containers

Replatform .NET Applications with Windows Containers: AWS Technical Guide

Table of Contents

Replatform .NET Applications with Windows Containers

Publication date: **January 4, 2022** (*Document history*)

Today, many architects, developers, and IT practitioners want to replatform their .NET Framework applications by moving them from Windows Virtual Machines (VMs) to Windows containers. This guide outlines a methodology to assess applications that are suitable to move to Windows containers, describes the business and technical benefits, and offers a prescriptive procedure using a sample application and reference architecture to guide organizations in the delivery of this process.

For comments, corrections, or questions, send an email to `<windowsmodernization@amazon.com>`.

Overview

Containers are becoming the primary technology for packaging and deploying applications. International Data Corporation (IDC) estimates that the number of container instances installed globally by enterprises (excluding public cloud providers) increased 133 percent year-over-year to 273.5 million instances in 2020, and is expected to grow five times to 1.39 billion instances by 2023. Within this enterprise containers install base, Windows container instances accounted for 19.5 million instances (7.1 percent of total) in 2020, and is expected to grow 8 times to 156.9 million instances (11.2 percent of total) by 2023. (Source: IDC Container Infrastructure Software Market Assessment: x86 Containers Forecast, 2018-2023, Doc# US46185620, April 2020.) The trend is clear that enterprises are adopting and standardizing the use of containers.

Before exploring how to complete the replatforming of existing .NET applications to Windows containers, this guide examines the reasons driving enterprises toward this approach.

The Windows operating system (OS) preexisted container technology. There were fundamental pieces of the OS that required changes to enable Windows to run in containers. Because of this, Windows containers were not introduced until 2016, three years after Docker released Linux containers. Kubernetes only recently debuted support for running Windows containers as worker nodes in April 2019.

Because of these releases, organizations have been moving their existing .NET Framework applications to Windows containers to improve their development agility, optimize the utilization of their infrastructure resources, increase their application portability across environments, and control the boundaries of their applications with container isolation. Windows containers also created an opportunity for organizations to migrate their existing .NET applications running on end-of-life versions, such as Windows Server 2003 and Windows Server 2008, to newer Windows Server versions with minimal code changes.

However, the benefits of Windows containers do not come without challenges. Existing development and IT teams that are well versed in building and running Windows-based applications today may lack hands-on experience with containers. This requires teams to develop new skillsets, operational techniques, and architectural designs to successfully deploy container-based applications in production. For example, when moving to Windows containers, teams must choose their Windows container type, runtime version, and orchestration engine to use. Teams must understand the technical nuances of Windows containers that require workarounds because of their early state of maturity. Additionally, new organizational practices, such as DevOps, are recommended to take full advantage of the automation that containers facilitate. All of these pieces contribute to the complexity that organizations face when adopting containers and this guide aims to demystify this picture to help teams adopt Windows containers where it makes sense for their technology stack.

Before you begin

Understand your drivers

Before you begin, take the time to understand your business and technical drivers and work backwards from your desired results to form a plan of action. Common business and technical drivers that motivate the approach to replatforming existing .NET Framework applications to Windows containers are outlined in the following tables.

Table 1 — Business drivers

Driver	Description	Solution
Accelerate innovation	Development and IT teams spend most of their time maintaining existing applicati ons rather than innovating.	Adopt containers to facilitat e DevOps practices and automation.
Lower total cost of ownership (TCO)	Infrastructure is underutilized and operational overhead slows teams down.	Move to containers to optimize resource utilization, and invest in automation to reduce manual processes for operations.
Address skills gap	Internal expertise in cloud-native technologies creates barriers to cloud adoption and modernization.	Use Amazon Web Services (AWS) official trainings and experts to upskill your staff.
Standardize technology	Different teams are using different tools which creates inefficiencies and prevents standardization.	Adopt containers for both Linux and Windows workloads to harmonize technology investments and internal knowledge base.

Table 2 — Technical drivers

Driver	Description	Solution
Increase productivity	Manual processes slow teams down and cause delays in release cycles.	Adopt DevOps practices and automation.
Scale with traffic demands	Applications are provisioned for peak load and underutilize infrastructure resources.	Use container auto scaling and stateless applications to scale up and down as traffic demands.
Predictable performance across environments	Application behavior is not consistent across environme nts leading to unforeseen problems.	Use containers for isolation and consistent behavior across environments.

If these drivers resonate with you and your teams, then replatforming your .NET Framework applications to Windows containers might be a good approach to optimize your existing Windows workloads. There are many options for modernizing your existing .NET Framework applications that also include refactoring to the cross-platform .NET 5+ and Linux, breaking down monoliths to microservices, and rewriting the application to be event-driven with serverless functions and AWS Lambda. Although these other approaches have their own unique set of benefits, replatforming to Windows containers is a common approach to optimize existing deployments, introduce cloud-native technologies to your teams, prepare for migrations, and to harden security.

Choosing container orchestration

As you replatform your application, you can select a container orchestrator that is most suitable for your requirements. 80 percent of all containers in the cloud run on AWS, and customers have a broad set of options to run and manage their containers on AWS. When choosing your container orchestration option, start with the question "How much of the container infrastructure do you want to manage?"

- **Amazon Elastic Container Service (Amazon ECS) on 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 Amazon Elastic Compute Cloud (Amazon EC2) instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers.

This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

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 identity and access management policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

- **Amazon Elastic Kubernetes Service (Amazon EKS)** — Amazon EKS is a managed service that allows you to run Kubernetes on AWS without needing to install and operate your own Kubernetes control plane or worker nodes. Amazon EKS provisions and scales the Kubernetes control plane, including the API servers and backend persistence layer, across multiple AWS Availability Zones for high availability and fault tolerance. Amazon EKS integrates with various AWS services such as Elastic Load Balancing (ELB), AWS Identity and Access Management (IAM), Amazon Virtual Private Cloud (Amazon VPC), and AWS CloudTrail to provide scalability and security for your applications.
- **Amazon Elastic Container Service (Amazon ECS)** — Amazon ECS is a highly scalable, high-performance container management service that supports Docker containers and allows you to run applications on a managed cluster of Amazon EC2 instances. With simple API calls, you can launch and stop container-enabled applications, query the complete state of your cluster, and access many familiar features like security groups, ELB, Amazon Elastic Block Store (Amazon EBS) volumes, and IAM roles. You can use Amazon ECS to schedule the placement of containers across your cluster based on your resource needs and availability requirements.
- **Self-managed containers on Amazon EC2** — EC2 virtual machines give you full control of your server clusters and provide a broad range of customization options. You can choose Amazon EC2 to run your container orchestration if you need full control over the installation, configuration, and management of the container orchestration environment.

Tools and libraries

AWS has many tools available for developers and IT practitioners to build and run container applications and infrastructure. The following table covers some of the primary tools that are useful for the replatforming process. You will use many of these tools and Amazon ECS in the *Walkthrough* section of this guide. For the latest news and releases, refer to the .NET on AWS landing page.

Table 3 – Tools and libraries

Tool or library	Description
AWS SDK for .NET	Use AWS services with purpose-built .NET libraries and APIs
AWS Cloud Development Kit (AWS CDK)	Define AWS infrastructure directly in code with the Cloud Development Kit for .NET
AWS Toolkit for Visual Studio	Extension for Visual Studio to create, debug, and deploy applications on AWS
AWS App2Container	Command-line tool to automate packaging applications into container images
AWS Porting Assistant for .NET	An analysis tool to assist with porting .NET Framework applications to .NET Core
AWS Extensions for .NET CLI	Build .NET Core applications with the .NET command-line interface (CLI)
AWS Tools for PowerShell	Manage AWS services and resources with PowerShell
AWS Toolkit for Azure DevOps	Extension for Azure DevOps to deploy applications on AWS
AWS CLI	Unified tool to manage your AWS services
AWS Copilot CLI	Simple declarative set of commands to deploy containers

Best practices

Choosing a Windows Server version

There are two primary release channels available for Windows Server 2019: the Long-Term Servicing Channel and the Semi-Annual Channel.

- **Long-Term Servicing Channel (LTSC)** — This is the release model that most customers use in production (formerly called the *Long-Term Servicing Branch*) where a new major version is released every 2–3 years. With LTSC, there are five years of mainstream support and five years of extended support.

  This channel is appropriate for systems that require a longer servicing option and functional stability. Deployments of Windows Server 2019 and earlier versions of Windows Server are not affected by the Semi-Annual Channel releases. The LTSC receives ongoing security and non-security updates, but it does not receive new features and functionality.
- **Semi-Annual Channel** — The Semi-Annual Channel is for customers who are innovating quickly to take advantage of new operating system capabilities at a faster pace, especially for containers and microservices. Windows Server products in the Semi-Annual Channel have new releases available twice a year, in spring and fall.

  Each release is supported for 18 months from the initial release. Most of the Semi-Annual Channel features are rolled into the next LTSC release of Windows Server. The editions, functionality, and supporting content might vary from release to release. In this model, Windows Server releases are identified by the year and month of release: for example, in 2017, a release in the ninth month (September) would be identified as version 1709.

Note

Microsoft has announced it is dropping Semi-Annual Channel (SAC) releases for Windows Server. Starting with Windows Server 2022 there will be only one release on the LTSC. It will get 10 years' support (five years mainstream, and five years extended).

Treat container instances as ephemeral servers

Windows administrator and IT professionals are responsible for several tasks that include OS patching, managing backups, restoring tests, and maintaining a healthy state for the applications they manage. These tasks change when using Windows containers and Amazon ECS because the containers should not be treated as an ordinary Windows Server instances. Rather, they should be treated as ephemeral instances that can be frequently removed, added, and replaced.

The Amazon ECS Windows container instance's sole purpose is to run containers, which drastically reduces management tasks compared to a Windows Server directly hosting an application. For example, when an Amazon ECS cluster is created, an AWS Auto Scaling group for the cluster is automatically created, which can be used to scale-in and scale-out the cluster based on the application's capacity requirements. Additionally, EC2 Image Builder can be used to automate the creation and deployment of container images so that the images remain up to date, consistent, and secure across the cluster. For details, refer to Building your own Amazon ECS–optimized Windows AMI.

Use multi-stage builds for container images

One of the most challenging parts to building container images is constraining the image size. Each instruction in the Dockerfile adds a layer to the image, and you need to clean up any unneeded artifacts before moving on to the next layer. To write an efficient Dockerfile, you have traditionally needed to employ tricks and other logic to keep the layers as small as possible and to ensure that each layer has the artifacts it needs from the previous layer and nothing else.

A common practice referred to as the *builder pattern* is to have one Dockerfile for development and a slimmed-down Dockerfile for production that only contains your application and the minimum dependencies required to run it. However, maintaining multiple Dockerfiles introduces surface area for error and adds to the maintenance overhead.

For example, in the following Dockerfile, the first block builds the .NET application and the second block uses the resulting image to build the Windows container image. This process is referred to as multi-stage builds.

```
FROM mcr.microsoft.com/dotnet/framework/sdk:4.8 AS build
WORKDIR /app
```

```
# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetmvcapp/*.csproj ./aspnetmvcapp/
COPY aspnetmvcapp/*.config ./aspnetmvcapp/
RUN nuget restore

# copy everything else and build app
COPY aspnetmvcapp/. ./aspnetmvcapp/
WORKDIR /app/aspnetmvcapp
RUN msbuild /p:Configuration=Release -r:False

FROM mcr.microsoft.com/dotnet/framework/aspnet:4.8 AS runtime
WORKDIR /inetpub/wwwroot
COPY --from=build /app/aspnetmvcapp/. ./
```

For more information on Docker best practices, refer to Docker development best practices and Best practices for writing Dockerfiles.

Caching layer strategy

Windows container images are large, ranging from 3.8 GB on disk for a Windows container image containing .NET framework based on Windows Server SAC edition to 5.1 GB on Windows Server 2019 LTSC. It's essential to implement a Windows container image layer caching strategy when utilizing EC2 Auto Scaling groups to avoid delays during task launch.

A common strategy is to pre-populate container images on the Amazon Machine Image (AMI) used by the Auto Scaling group to avoid two time-expensive Docker operations:

- Downloading the image from the repository
- Extracting the image as layers on the local operating system

The download and extraction phase consumes sequential I/O operations per second on the disk that directly impacts the container instance's performance until all images are downloaded and extracted. Also, it imposes a delay on the container's readiness to receive traffic, because the process can take two to five minutes to complete, depending on the size of the image.

The Amazon ECS container instances should be treated as ephemeral servers. There is an option to use EC2 Image Builder to build your own AMI with all of the necessary patches and security configuration. This service also enables you to include an additional step in the EC2 Image Builder pipeline to download and extract Docker images directly on the AMI, which reduces the time it takes to launch an EC2-based task on your Amazon ECS cluster.

For more information on caching layer strategy, refer to Speeding up Windows container launch times with EC2 Image Builder and image cache strategy.

Cost considerations

Cloud computing

With cloud computing, companies have access to a scalable platform, low-cost storage, database technologies, and tools on which to build enterprise-grade solutions. Cloud computing helps businesses reduce costs and complexity, adjust capacity on-demand, accelerate time to market, increase opportunities for innovation, and enhance security.

Weighing the financial considerations of operating an on-premises data center compared to using cloud infrastructure is not as simple as comparing hardware, storage, and compute costs. Whether you own your own data center or rent space at a colocation facility, you have to manage investments that include capital expenditures, operational expenditures, staffing, opportunity costs, licensing, and facilities overhead.

AWS pricing model

AWS offers a simple, consistent, pay-as-you-go pricing model, so you are charged only for the resources you consume. With this model, there are no upfront fees, minimum commitments, and long-term contracts required. There is also flexibility to choose the pricing model that best fits your needs if the pay-as-you-go model is not optimal for your use case. Short descriptions of all of these pricing models follow.

- **On-Demand Instance** — With On-Demand Instances, you pay for compute capacity by the second with no minimum commitments required.
- **Savings Plans** — A flexible pricing model offering lower prices compared to On-Demand pricing, in exchange for a specific usage commitment (measured by hourly rate) for a one-year or three-year period. AWS offers three types of Savings Plans:
  - Compute Savings Plans
  - EC2 Instance Savings Plans
  - Amazon SageMaker Savings Plans

  Compute Savings Plans apply to usage across Amazon EC2, AWS Lambda, and AWS Fargate.
- **Reserved Instance** — For longer-term savings, you can purchase in advance. In addition to providing a significant discount (up to 60 percent) compared to On-Demand Instance pricing, Reserved Instances allow you to reserve capacity.

- **Spot Instance** — You can bid for unused Amazon EC2 capacity. Instances are charged the Spot Price, which is set by Amazon EC2 and fluctuates, depending on supply and demand. For more information, refer to Amazon EC2 Spot Instances.

For more information, refer to the AWS Cloud Economics Center and Savings Plans.

AWS container services

There are several cost aspects to consider when running applications on AWS. These include but are not limited to storage, data transfer, service usage, compute, and operations. When considering replatforming to Windows containers, it's important to evaluate the resource utilization of the existing application. One of the primary benefits of containers is the ability to *bin-pack* several containerized application instances on a reduced footprint of infrastructure. This leads to improvements in resource utilization which can result in considerable cost savings.

A common concern when considering a replatforming effort is the cost of changing application code or introducing new DevOps processes needed to build and deploy the application. When targeting Windows containers, application developers can minimize the number of changes needed for this new environment, and in many cases find that there are no changes required to the code itself. Modifying CI/CD processes to deliver container images as the build artifact as opposed to traditional application binaries require changes, but has the benefit of removing the step of installing and configuring code in the target environment; instead, the container image can be fully configured at build time and tested as part of the build pipeline prior to delivery.

Finally, there is an additional cost component that is frequently overlooked which is managing the server infrastructure and application deployment. In a traditional on-premises environment, application servers must be configured to run the application as part of the application deployment process; this can be a time-consuming process, and dependencies may differ from application to application.

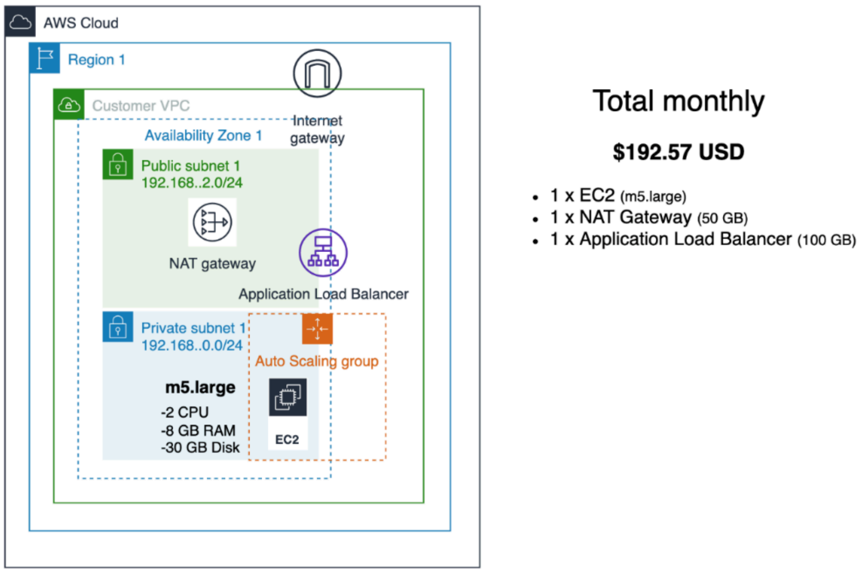
The fact that the application container image delivers the application dependencies and application code as a single, pre-packaged build artifact dramatically simplifies the deployment process and time needed for any troubleshooting. Additionally, this delivery mechanism decouples the configuration and patching of servers from the run environment, and allows the operations management team to have a single, unified process for managing the server infrastructure, which reduces management overhead.

Cost comparison

There are four different services that AWS provides to run Windows container-based applications (containers on Amazon EC2, Amazon EKS, Amazon ECS, and AWS Fargate). To understand the cost implications of running containers on each of these services, this guide presents an example of a simple application architecture and compares the cost of running it on each service with an On-Demand pricing model in the `us-east-1` Region. You can choose a different pricing structure such as Spot Instances or Saving Plans, which are supported for the services covered in this guide. The examples in the following sections were generated by the AWS Pricing Calculator.

This guide focuses on compute (where the containers run) and operations cost (staffing) of managing the compute resources. It does not include the cost consolidation of *bin-packing* applications or of running a database to simplify the analysis.

As the baseline for the comparison, this guide uses an application running on Windows in Amazon EC2, as shown in the following diagram.



Total monthly

$192.57 USD

- 1 x EC2 (m5.large)
- 1 x NAT Gateway (50 GB)
- 1 x Application Load Balancer (100 GB)

Legacy .NET application running on EC2 Windows

Summary by service

The following table represents the summary of running the above application on each AWS container service. These figures include the cost of compute, the Amazon VPC, and ELB. You can see that running on Linux costs less than running on Windows for each service which is influenced by the license-included cost of Windows. Additionally, the cost for Amazon EKS compared to the other options is higher due to the per-cluster cost of cluster management.

Table 4 – Monthly cost summary by service

Service	Monthly cost
Amazon EC2	$192.57 (details)
Amazon EKS	$265.57 (details)
Amazon ECS	$192.57 (details)
AWS Fargate	$124.41 (details)

Self-managed containers on Amazon EC2

Running self-managed containers on Amazon EC2 gives you the highest level of control over the underlying compute but it comes with the highest TCO as you manage the entirety of the container's lifecycle. Additionally, you are responsible for optimally utilizing the underlying compute, rather than leaving this to the managed container orchestrator. For more information, refer to the details links in the preceding table.

- **Amazon EKS** — With Amazon EKS, you pay $0.10 per hour for each cluster that you create. You can use a single Amazon EKS cluster to run multiple applications by taking advantage of Kubernetes namespaces and IAM security policies. You can run Windows containers on Amazon EKS using the Amazon EC2 launch type and on-premises using AWS Outposts or Amazon EKS Anywhere.

  If you are using Amazon EC2 (including with Amazon EKS–managed node groups), you pay for AWS resources (such as EC2 instances or EBS volumes) you create to run your Kubernetes worker nodes. You only pay for what you use, as you use it; there are no minimum fees and no upfront commitments.

  The calculations in this guide use Amazon EC2 for compute. For more details on Amazon EKS pricing, refer to the Amazon EKS pricing webpage.
- **Amazon ECS** — With Amazon ECS, there is no additional charge for the cluster orchestrator. You can run Windows containers on Amazon ECS using the Amazon EC2 launch type and on-premises using AWS Outposts or Amazon ECS Anywhere. Again, you only pay for what you use, as you use it; there are no minimum fees and no upfront commitments.

The calculations in this guide use EC2 for compute. For more details on ECS pricing, refer to the Amazon ECS pricing webpage.

- **AWS Fargate** — With AWS Fargate, there are no upfront costs and you pay only for the resources you use. You pay for the amount of vCPU, memory, and storage resources consumed by your containerized applications running on Amazon ECS or Amazon EKS.

  The calculations in this guide use Fargate for compute. For more details on Fargate pricing, refer to the AWS Fargate pricing webpage.

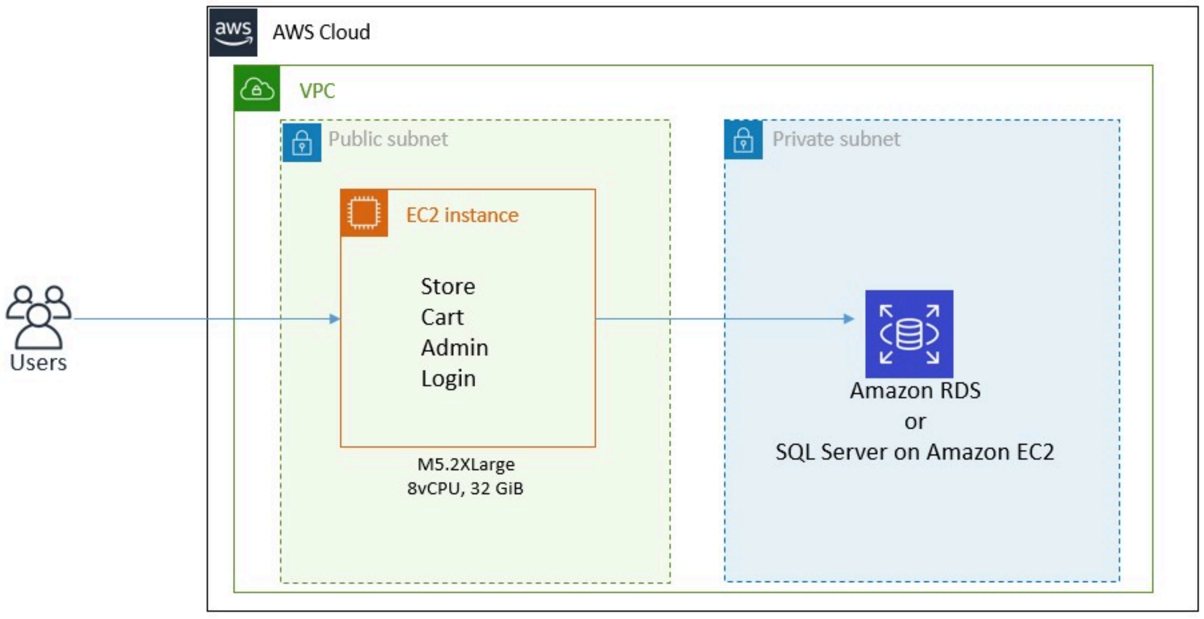
Operational costs (staffing)

One of the benefits of using managed services is that you can save time by not having to perform operations that are considered undifferentiated heavy lifting. Managed services in AWS such as Amazon ECS and Amazon EKS remove the burden of managing your container orchestrator, freeing your resources to focus more on building your applications to drive business outcomes.

Architecture overview

It’s common for enterprise .NET applications built in the last decade to follow a layered n-tier architectural approach. The functionality for different aspects of the application is logically separated yet bundled together as interdependent code modules. There are multiple advantages of an n-tier architecture. They’re easy to develop, more feasible to test if the application size is small, and can scale vertically. However, as more functionality is added and the code base grows, the applications become cumbersome to manage, change, and scale. State-dependent applications are particularly difficult to scale horizontally, and the compute capacity must be provisioned to consider the peak load.

This guide uses the familiar MvcMusicStore reference application, which is built on an n-tier approach using ASP.NET MVC and .NET Framework. It maintains a session state in memory, and it can scale vertically by default. The data persistence layer uses Microsoft SQL Server. The high-level architecture for this application follows.



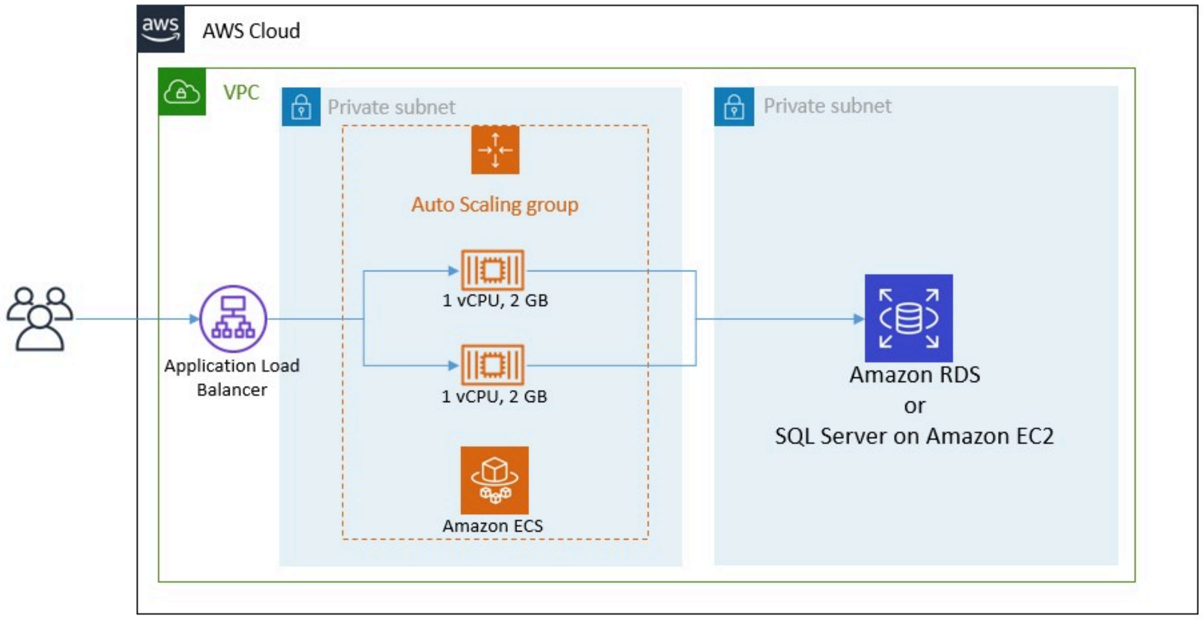
MvcMusicStore legacy architecture

Many customers start their cloud journey with a lift-and-shift approach, running their n-tier .NET Framework applications on EC2 without any code changes. It’s common for these deployments to have more than one EC2 Windows instance with an Application Load Balancer, routing the user requests to one of the EC2 instances. A stateful application can have session affinity (sticky sessions) enabled at the Application Load Balancer level to create an affinity between a client and a specific EC2 instance.

Along with the Application Load Balancer, developers can use AWS Auto Scaling to monitor an application and automatically adjust capacity to maintain steady, predictable performance at the lowest possible cost. Amazon RDS for SQL Server is a managed database service that frees you up to focus on application development by managing time-consuming database administration tasks, including provisioning, backups, software patching, monitoring, and hardware scaling.

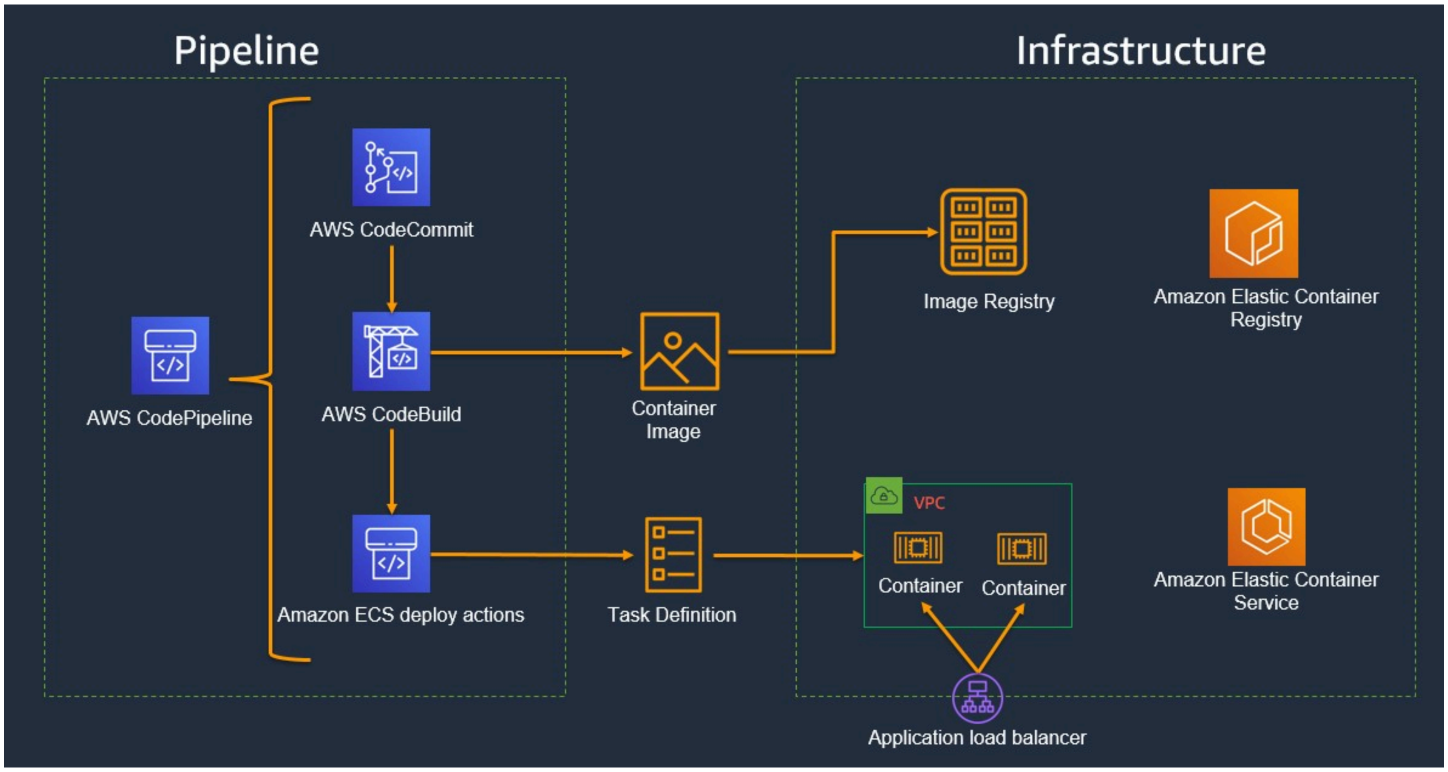
In this guide, you replatform this traditional n-tier .NET Framework application to run on Amazon ECS using AWS App2Container. As mentioned in the previous sections, running self-managed containers on Amazon EC2 and Amazon EKS is also an option, but Amazon ECS is a simple yet powerful place to start if you and your teams are new to containers.

Amazon ECS runs natively with AWS services such as Application Load Balancer and Auto Scaling, allowing developers to start with the minimum amount of compute to meet user requirements and scale dynamically as the incoming traffic increases. The high-level architecture for the containerized version of this application follows, and is the target of the transformation detailed in this guide. In the following diagram, the 1 vCPU, 2 GB blocks represent the Amazon ECS tasks where the application containers run.



MvcMusicStore architecture on Amazon Elastic Container Service

Additional benefits of moving to containers are realized when teams also implement automation and DevOps. A cloud-optimized, containerized application allows you to quickly and frequently deliver consistent applications to your users. A common development pipeline for nearly continuous deployment with AWS follows.

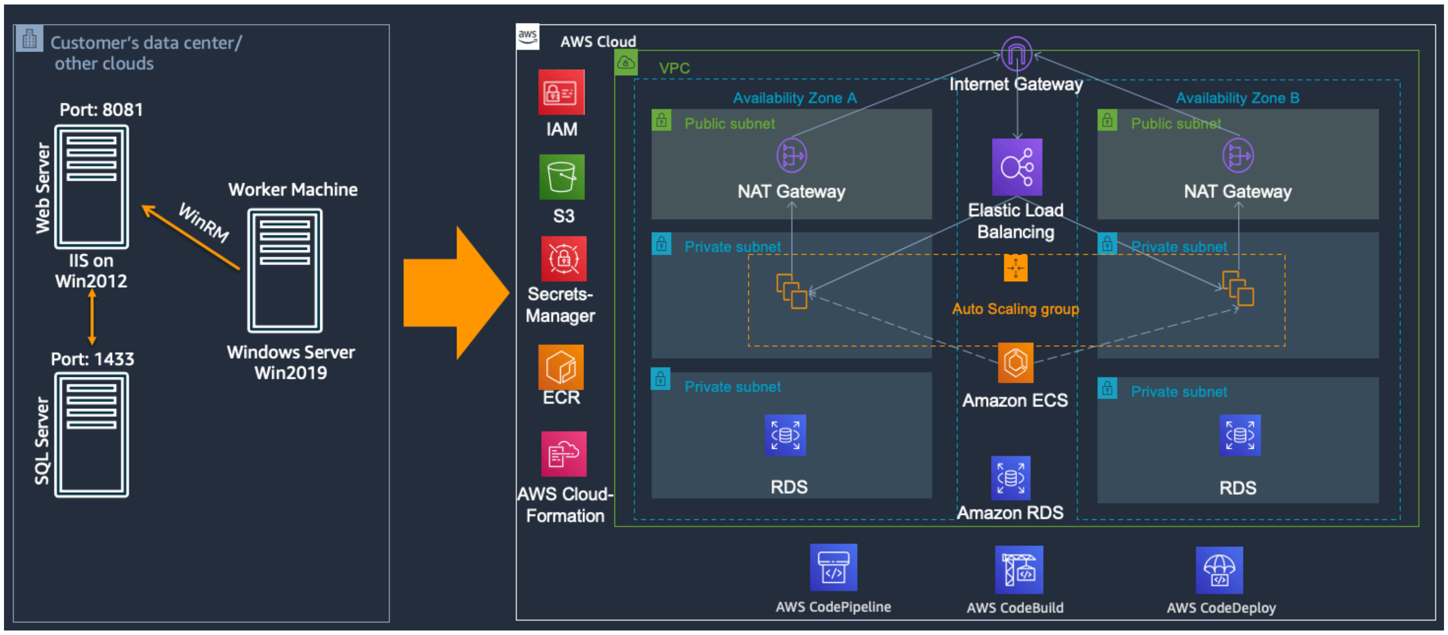


A common development pipeline for nearly continuous deployment with AWS

Now that you have an idea of the before and after state of the application that you will modernize in this guide, walk through the procedural guidance to complete this transformation quickly and safely.

Walkthrough

This walkthrough uses the MvcMusicStore application to demonstrate how to replatform an ASP.NET MVC and Entity Framework-based application that runs on Internet Information Services (IIS) to Windows containers. You will use a tool named AWS App2Container to containerize and deploy it. This tool is provided free of charge and it automates many of the steps necessary to convert an existing VM-based IIS application to one hosted in a container. Additionally, the tool can be used to deploy the containerized application to AWS and create a CI/CD pipeline so that changes to the application can be reliably pushed to the deployment environment. This walkthrough focuses on only the application component (not the database migration) of the following architecture.



MvcMusicStore application architecture

For this walkthrough, there are three stages:

1. **Prerequisites** — Prepare the AWS environment for replatforming.
2. **Containerization** — Use App2Container to analyze and containerize the app.
3. **Deployment** — Use App2Container to deploy the containerized app to Amazon ECS and create a CI/CD pipeline.

Note

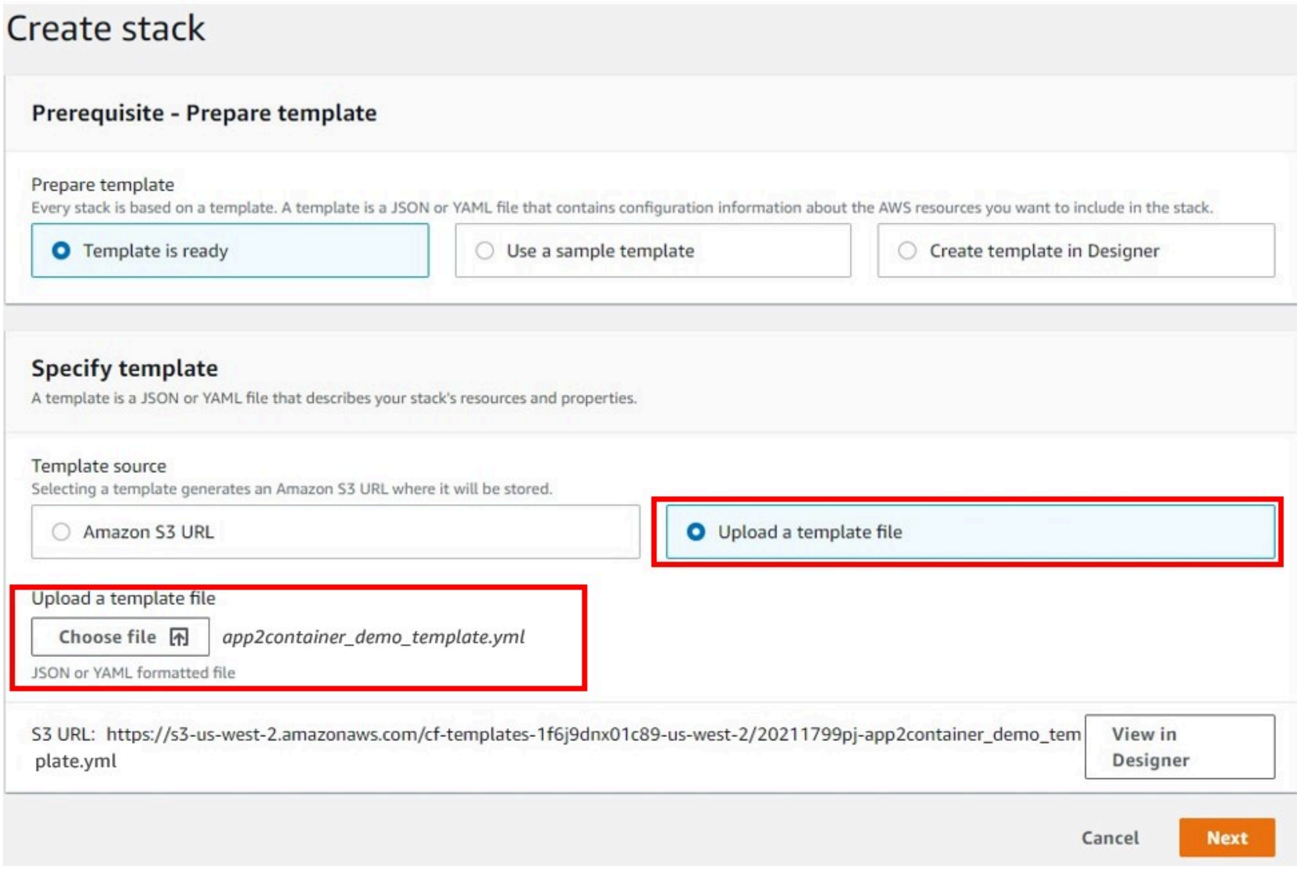
Do not forget to end your AWS resources upon completion of the walkthrough. This can be accomplished by going to the CloudFormation service in the AWS Management Console and deleting each stack.

Topics

Prerequisites

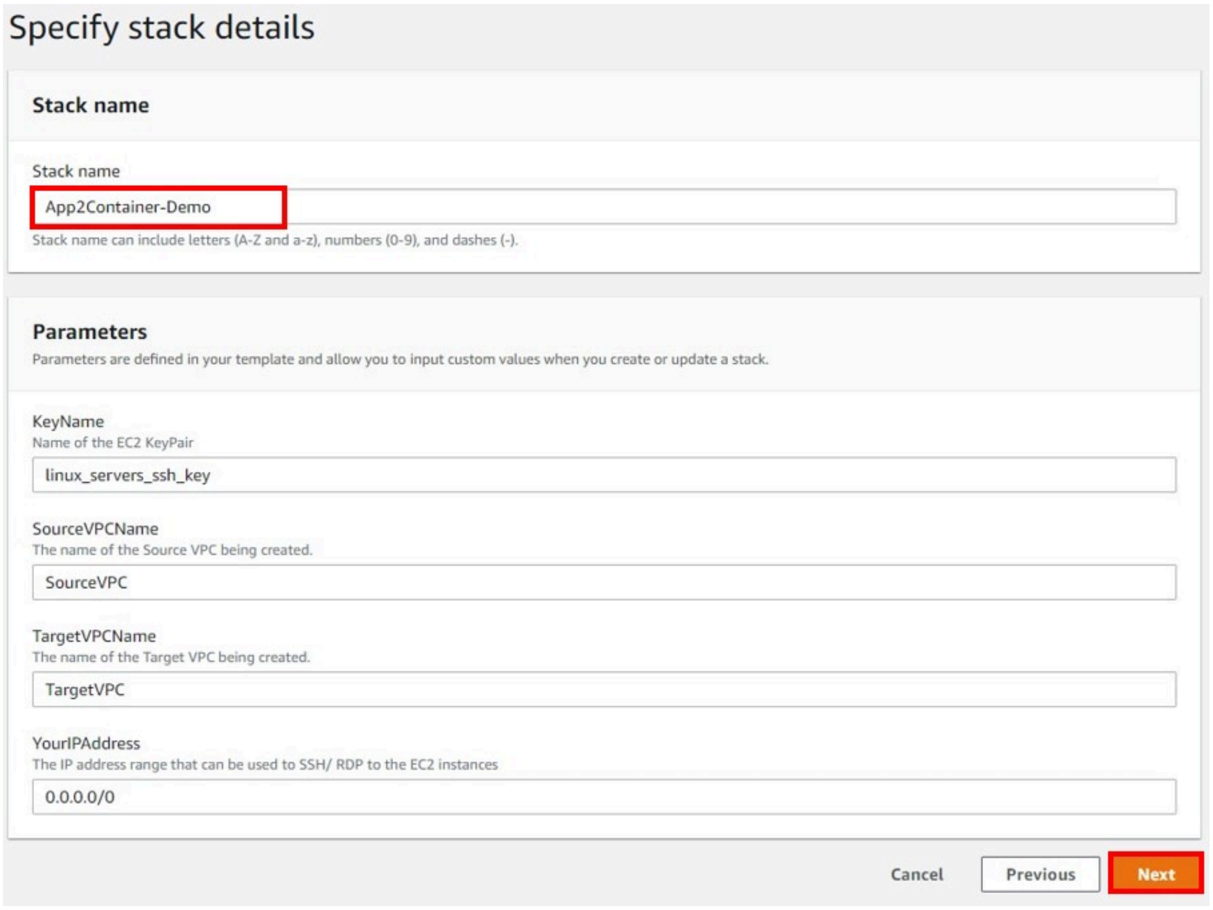
Deploy the AWS environment

1. If you do not have an AWS account, follow the How do I create and activate a new AWS account? article to create one that you will use throughout this walkthrough.
2. Download the App2Container demo CloudFormation template to your local machine for the AWS environment that you will deploy for the demo environment.
3. Deploy the AWS environment with CloudFormation (sign-in required).
4. In the **Create stack** dialog box in **Specify template**, choose Upload a template file, and select the `app2container_demo_template.yml` file that you downloaded to your local machine. Choose **Next**.


Create stack
Prerequisite - Prepare template
Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.
● Template is ready
○ Use a sample template
○ Create template in Designer
Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.
Template source
Selecting a template generates an Amazon S3 URL where it will be stored.
○ Amazon S3 URL
● Upload a template file
Upload a template file
Choose file
app2container_demo_template.yml
JSON or YAML formatted file
S3 URL: https://s3-us-west-2.amazonaws.com/cf-templates-1f6j9dnx01c89-us-west-2/20211799pj-app2container_demo_template.yml
View in Designer
Cancel
Next

Create stack *dialog box*

5. In the **Specify stack details** dialog box, enter **`App2Container-Demo`** as the stack name. You can optionally add your public IP address in the **YourIPAddress** field to limit the access only to your IP address. The default value (0.0.0.0/0) will allow traffic for all IP addresses. Choose **Next**.



Specify stack details

Stack name

Stack name

App2Container-Demo

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

KeyName
Name of the EC2 KeyPair
linux_servers_ssh_key

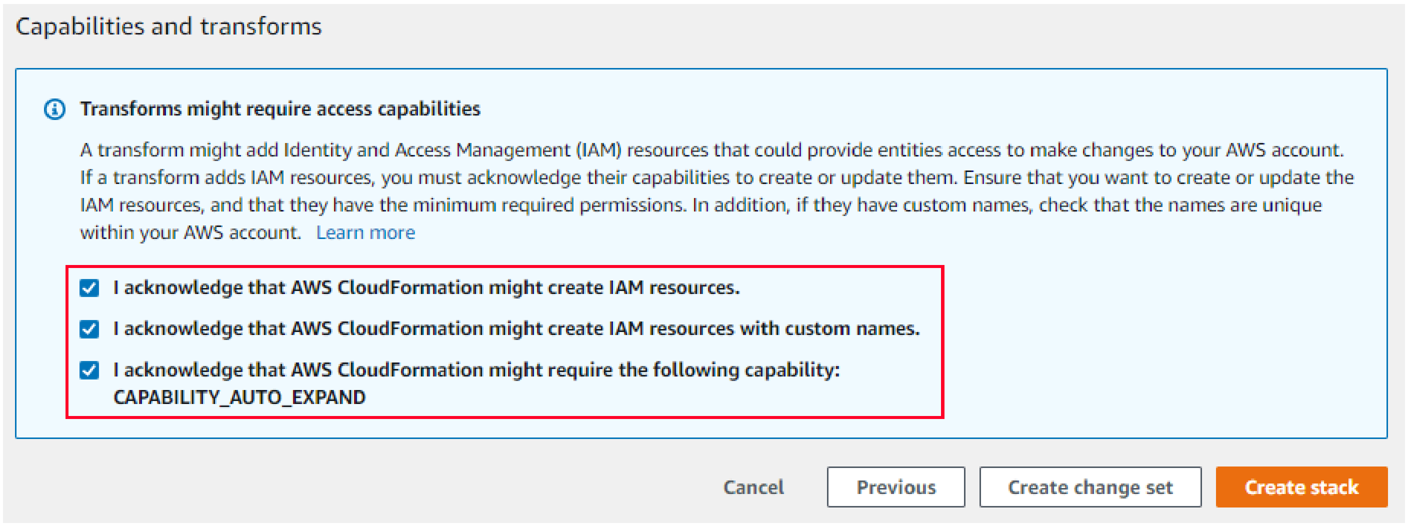
SourceVPCName
The name of the Source VPC being created.
SourceVPC

TargetVPCName
The name of the Target VPC being created.
TargetVPC

YourIPAddress
The IP address range that can be used to SSH/ RDP to the EC2 instances
0.0.0.0/0

Specify stack details *dialog box*

6. In the **Configure stack options** dialog box, don’t make any changes, and choose **Next**.
7. In the **Capabilities and transforms** dialog box, review, scroll to the bottom of the page, and check all check boxes. Choose **Create stack**.



Capabilities and transforms

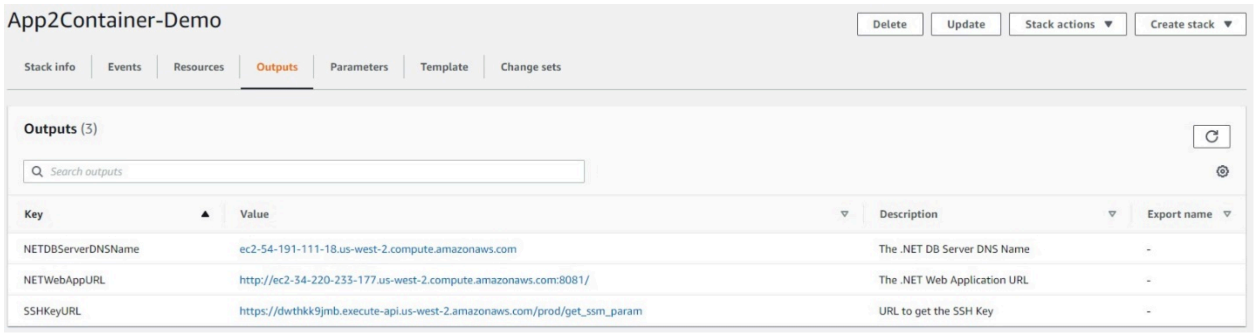
Transforms might require access capabilities

A transform might add Identity and Access Management (IAM) resources that could provide entities access to make changes to your AWS account. If a transform adds IAM resources, you must acknowledge their capabilities to create or update them. Ensure that you want to create or update the IAM resources, and that they have the minimum required permissions. In addition, if they have custom names, check that the names are unique within your AWS account. Learn more

- ✓ I acknowledge that AWS CloudFormation might create IAM resources.
- ✓ I acknowledge that AWS CloudFormation might create IAM resources with custom names.
- ✓ I acknowledge that AWS CloudFormation might require the following capability: CAPABILITY_AUTO_EXPAND

***Capabilities and transforms** dialog box*

- If the deployment fails, go to the deployment stack **Events** tab and confirm the root cause. A common root cause is `ecsExecutionRole` or `ecsAutoscaleRole` IAM roles already exist in the AWS account. Delete them and re-run deployment of the environment from the CloudFormation template.
- When the template is in the `CREATE_COMPLETE` status, you can find information about created source environment by going to **AWS Management Console > CloudFormation**, choosing **App2Container-Demo stack**, and choosing the **Outputs** tab.



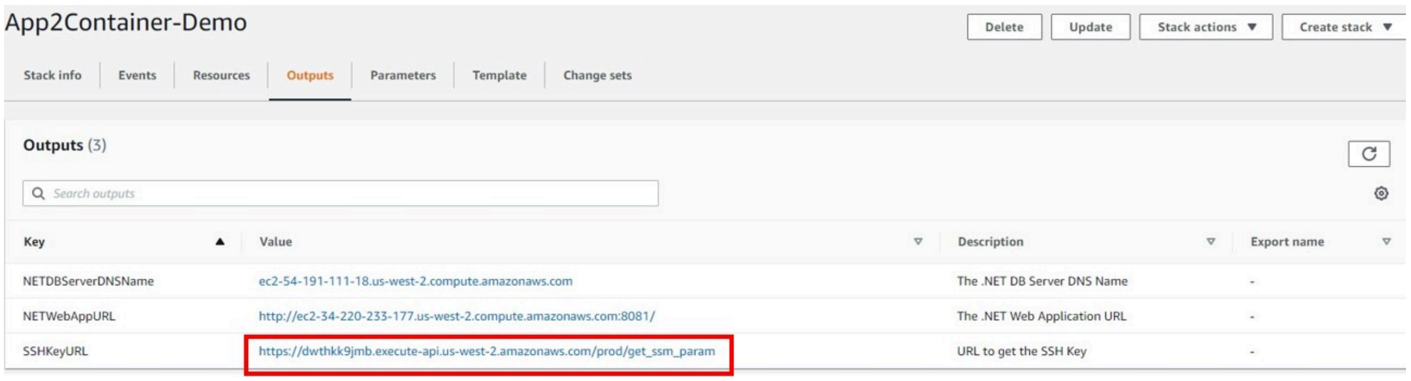
Key	Value	Description	Export name
NETDBServerDNSName	ec2-54-191-111-18.us-west-2.compute.amazonaws.com	The .NET DB Server DNS Name	-
NETWebAppURL	http://ec2-34-220-233-177.us-west-2.compute.amazonaws.com:8081/	The .NET Web Application URL	-
SSHKeyURL	https://dwthkk9jmb.execute-api.us-west-2.amazonaws.com/prod/get_ssm_param	URL to get the SSH Key	-

***Outputs** tab of the **App2Container-Demo** stack dialog box*

Connect to deployment

In the previous section, you deployed the stack for your environment that included three EC2 instances. The first EC2 instance is a Windows Server 2019 instance for the worker machine where you will run App2Container, the second is a Windows Server 2012 instance running the MvcMusicStore application, and the third is a Windows instance running SQL Server. Before you proceed to setting up App2Container, confirm that the deployment succeeded:

1. Go to **CloudFormation** in the AWS Management Console and select the **App2Container-Demo** stack that you launched in the previous step.
2. On the **Outputs** tab, select the **SSHKeyURL** link. This brings you to the Secure Shell (SSH) key that will be used to retrieve Administrator access. Save the key to your local machine.

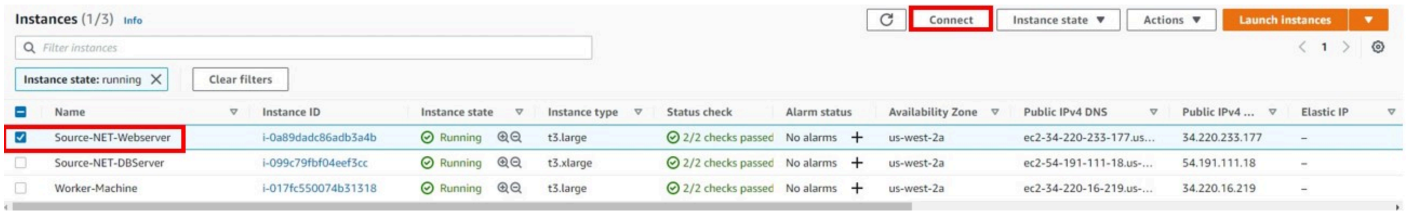


Key	Value	Description	Export name
NETDBServerDNSName	ec2-54-191-111-18.us-west-2.compute.amazonaws.com	The .NET DB Server DNS Name	-
NETWebAppURL	http://ec2-34-220-233-177.us-west-2.compute.amazonaws.com:8081/	The .NET Web Application URL	-
SSHKeyURL	https://dwthkk9jmb.execute-api.us-west-2.amazonaws.com/prod/get_ssm_param	URL to get the SSH Key	-

*Select the **SSHKeyURL** link*

3. Go the EC2 service in the AWS Management Console and navigate to your instances.

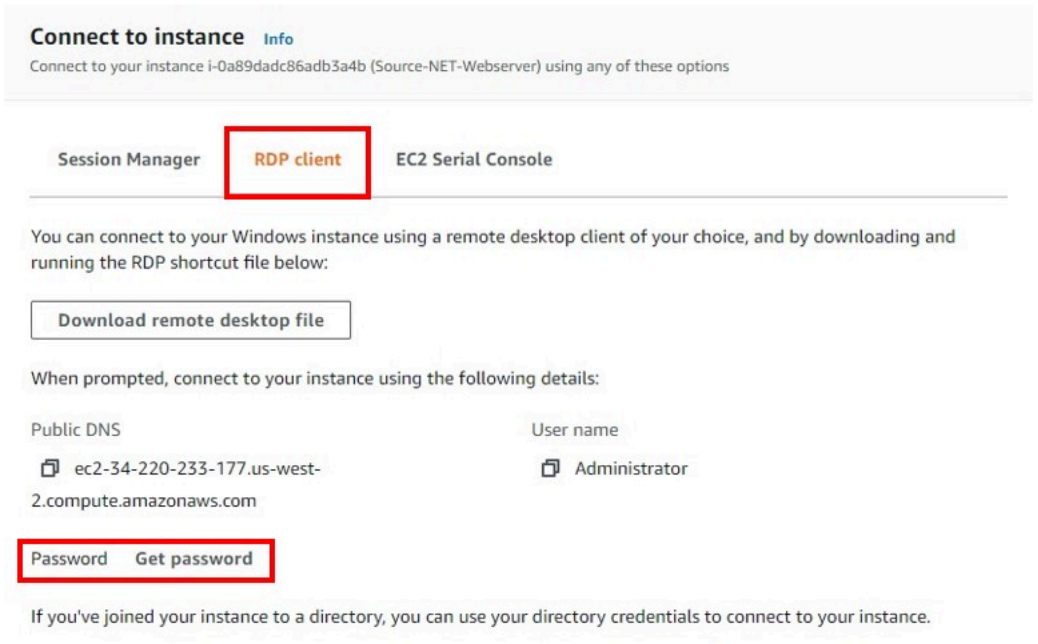
4. Select the **Source-NET-Webserver instance**, and choose **Connect** in the upper-right corner.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Source-NET-Webserver	i-0a89dadc86adb3a4b	Running	t3.large	2/2 checks passed	No alarms	us-west-2a	ec2-34-220-233-177.us...	34.220.233.177	–
Source-NET-DBServer	i-099c79fbf04eef3cc	Running	t3.xlarge	2/2 checks passed	No alarms	us-west-2a	ec2-54-191-111-18.us-...	54.191.111.18	–
Worker-Machine	i-017fc550074b31318	Running	t3.large	2/2 checks passed	No alarms	us-west-2a	ec2-34-220-16-219.us-...	34.220.16.219	–

*Select the **Source-NET-Webserver instance** and choose **Connect***

5. In the **Connect to instance** dialog box, choose the **RDP client** tab, and select **Get password**. A popup box appears.



Connect to your instance i-0a89dadc86adb3a4b (Source-NET-Webserver) using any of these options

You can connect to your Windows instance using a remote desktop client of your choice, and by downloading and running the RDP shortcut file below:

When prompted, connect to your instance using the following details:

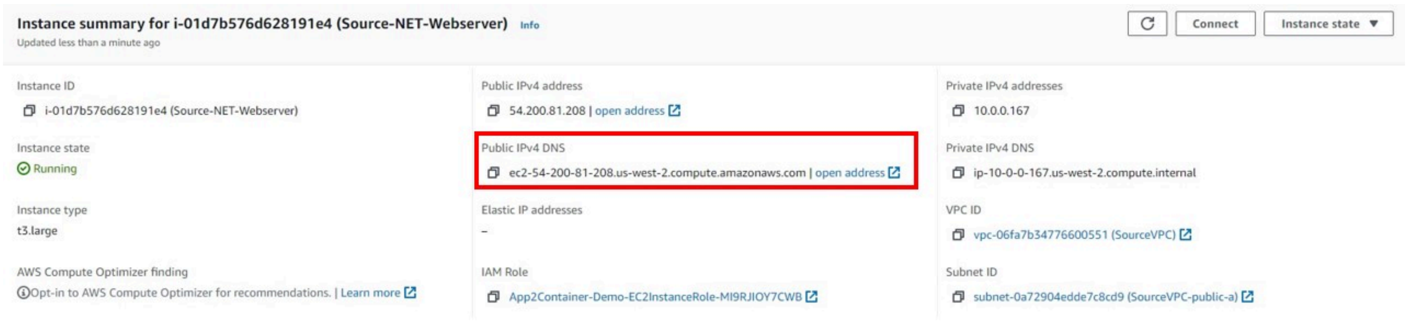
Public DNS	User name
ec2-34-220-233-177.us-west-2.compute.amazonaws.com	Administrator

Password Get password

If you've joined your instance to a directory, you can use your directory credentials to connect to your instance.

*Choose the **RDP client** tab and select **Get password***

6. In the **Get password** popup box, paste the SSH key that you saved to your local machine and select **Decrypt Password**. Copy the password that appears on the screen to your local machine. You will need it to log in to the remote machine through Remote Desktop Protocol (RDP).
7. Go back to your **Instances** view and copy the Public IPv4 DNS string for the web server instance to your local machine.

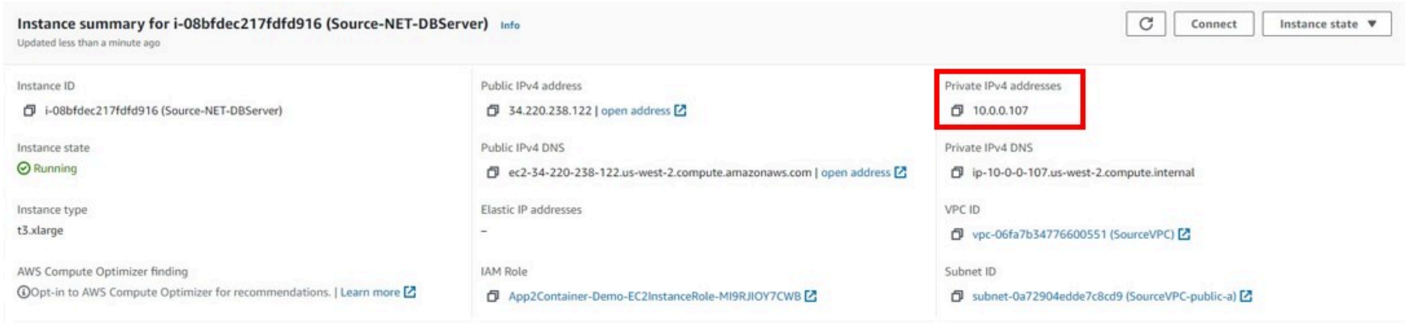


Instance summary for i-01d7b576d628191e4 (Source-NET-Webserver) Info
Updated less than a minute ago

Instance ID	Public IPv4 address	Private IPv4 addresses
i-01d7b576d628191e4 (Source-NET-Webserver)	54.200.81.208 | open address	10.0.0.167
Instance state	Public IPv4 DNS	Private IPv4 DNS
Running	ec2-54-200-81-208.us-west-2.compute.amazonaws.com | open address	ip-10-0-0-167.us-west-2.compute.internal
Instance type	Elastic IP addresses	VPC ID
t3.large	–	vpc-06fa7b34776600551 (SourceVPC)
AWS Compute Optimizer finding	IAM Role	Subnet ID
Opt-in to AWS Compute Optimizer for recommendations. | Learn more	App2Container-Demo-EC2InstanceRole-MI9RJIOY7CWB	subnet-0a72904edde7c8cd9 (SourceVPC-public-a)

*Copy the **Public IPv4 DNS** string for the web server instance to your local machine*

8. Go to your **DBServer** instance and copy the **Private IPv4** address to your local machine. You will need this while editing the `Web.config` file for the application to connect to the database.



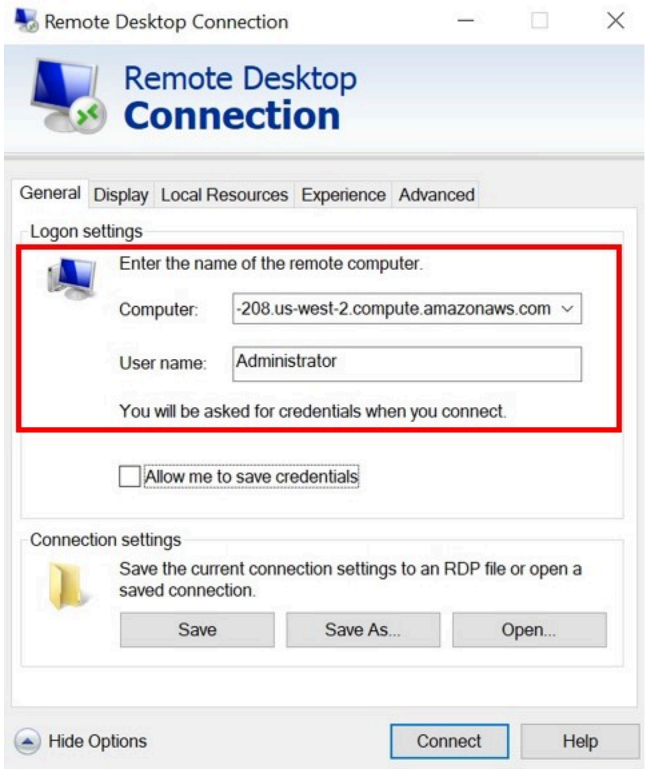
Instance summary for i-08bfdec217fdfd916 (Source-NET-DBServer) Info
Updated less than a minute ago

Instance ID	Public IPv4 address	Private IPv4 addresses
i-08bfdec217fdfd916 (Source-NET-DBServer)	34.220.238.122 | open address	10.0.0.107
Instance state	Public IPv4 DNS	Private IPv4 DNS
Running	ec2-34-220-238-122.us-west-2.compute.amazonaws.com | open address	ip-10-0-0-107.us-west-2.compute.internal
Instance type	Elastic IP addresses	VPC ID
t3.xlarge	–	vpc-06fa7b34776600551 (SourceVPC)
AWS Compute Optimizer finding	IAM Role	Subnet ID
Opt-in to AWS Compute Optimizer for recommendations. | Learn more	App2Container-Demo-EC2InstanceRole-MI9RJIOY7CWB	subnet-0a72904edde7c8cd9 (SourceVPC-public-a)

*Copy the **Private IPv4** address to your local machine*

9. Open Remote Desktop Connection on your local machine and paste the web server DNS string into the **Connection** field.

10Enter **Administrator** as the user name, and choose **Connect**.


Remote Desktop Connection
Remote Desktop Connection
General
Display
Local Resources
Experience
Advanced
Logon settings
Enter the name of the remote computer.
Computer:
-208.us-west-2.compute.amazonaws.com
User name:
Administrator
You will be asked for credentials when you connect.
Allow me to save credentials
Connection settings
Save the current connection settings to an RDP file or open a saved connection.
Save
Save As...
Open...
Hide Options
Connect
Help

*Enter **Administrator** as the user name and choose **Connect***

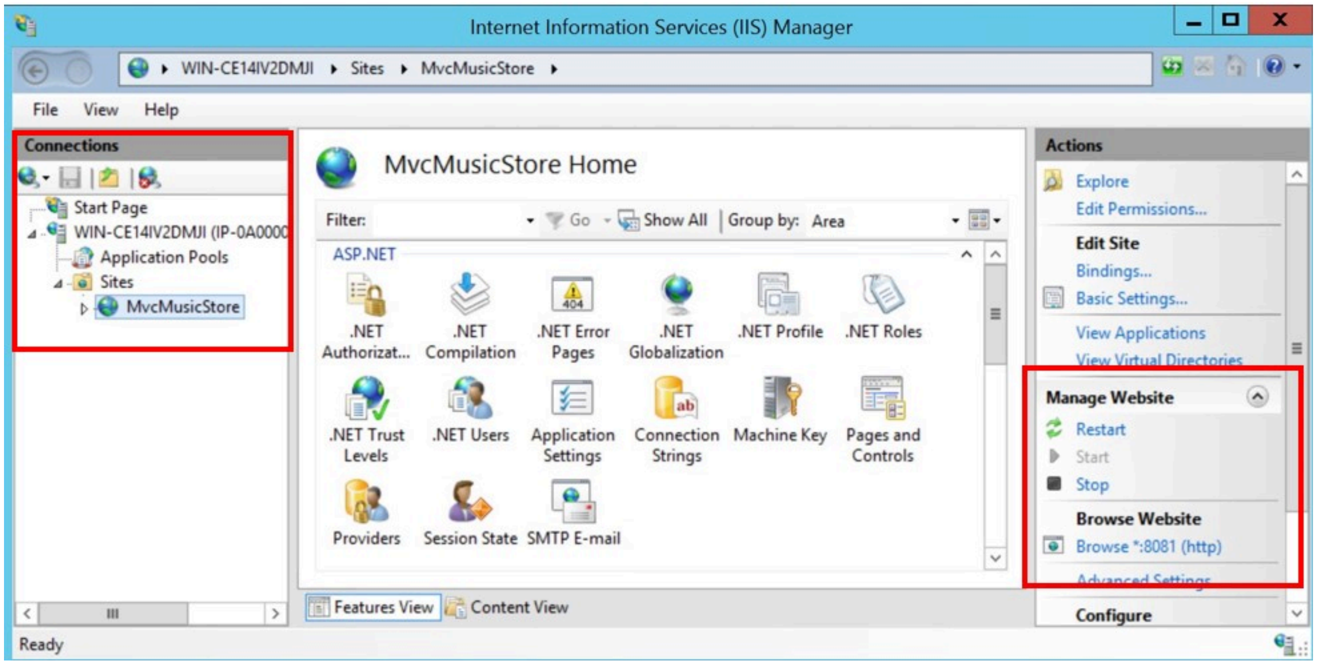
11. After you are connected to the web server instance, open PowerShell.

12. Run the following command to configure your application to connect to the database. Replace `<change-this-to-your-private-db-ip>` with the private IPv4 address of your database instance that you copied in step 8.

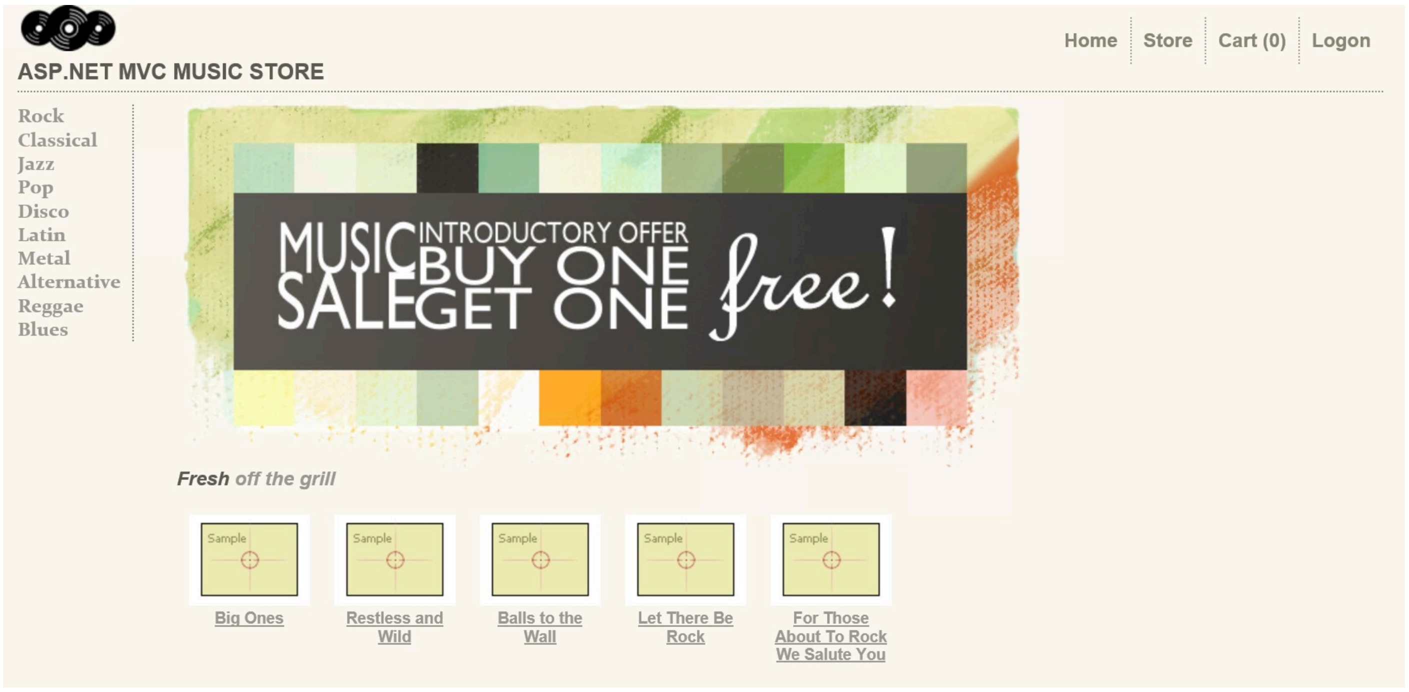
```
(Get-Content C:/dotnet-modernization-music-
store/MvcMusicStore/Web.config).replace('Data Source=.',
'Data Source=<change-this-to-your-private-db-ip>') | Set-Content
C:/dotnet-modernization-music-store/MvcMusicStore/Web.config
```

13. Go to **Administrative Tools > Internet Information Services (IIS) Manager**. Choose the **MvcMusicStore** application from the left pane.

14. Choose **Restart** in the right pane, and then choose **Browse 8081**. The MvcMusicStore user interface should appear in the internet browser.



Internet Information Services (IIS) Manager

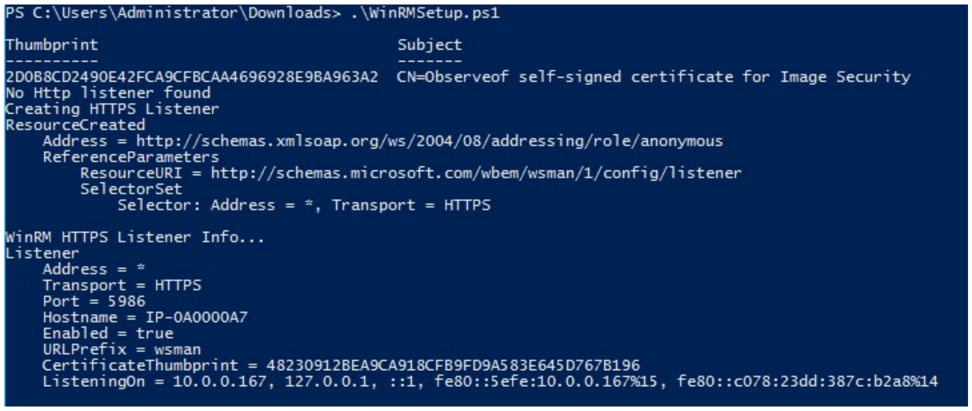


MvcMusicStore user interface

15. Now that you have confirmed that the application is successfully running in your environment, you need to install WinRM so that App2Container can remotely connect to your web server instance to analyze and containerize your application. Run the following in PowerShell:

```
cd Downloads
.\WinRMSetup.ps1
```

You should observe the following output:



```
PS C:\Users\Administrator\Downloads> .\WinRMSetup.ps1

Thumbprint                                Subject
----------                                -------
2D0B8CD2490E42FCA9CFBCAA4696928E9BA963A2  CN=Observeof self-signed certificate for Image Security
No Http listener found
Creating HTTPS Listener
ResourceCreated
    Address = http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    ReferenceParameters
        ResourceURI = http://schemas.microsoft.com/wbem/wsman/1/config/listener
        SelectorSet
            Selector: Address = *, Transport = HTTPS

WinRM HTTPS Listener Info...
Listener
    Address = *
    Transport = HTTPS
    Port = 5986
    Hostname = IP-0A0000A7
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint = 48230912BEA9CA918CFB9FD9A583E645D767B196
    ListeningOn = 10.0.0.167, 127.0.0.1, ::1, fe80::5efe:10.0.0.167%15, fe80::c078:23dd:387c:b2a8%14
```

PowerShell output

You have confirmed that your MvcMusicStore application is running on the web server instance on IIS. Additionally, you have set up your web server instance with WinRM so that App2Container can remotely access it from the worker machine. You are ready to move on to the next step to set up App2Container.

Set up App2Container prerequisites

App2Container needs access to AWS services to run most of its commands. There are two very different sets of permissions needed to run App2Container commands:

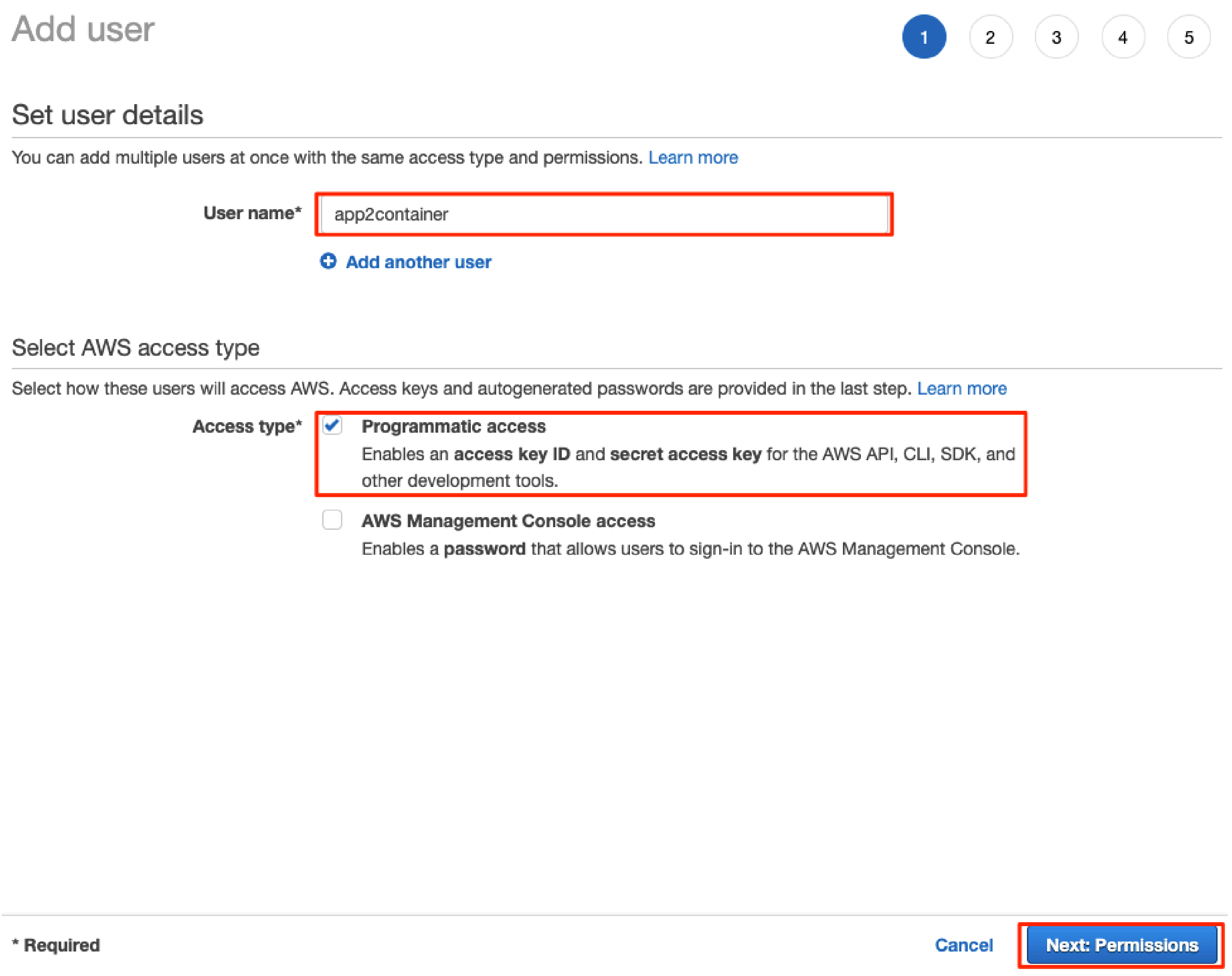
- The General Purpose user or group can run all of the commands except commands that are run with the `--deploy` option.
- For deployment, App2Container must be able to create or update AWS objects for container management services (Amazon ECR with Amazon ECS, Amazon EKS, or Fargate) and to create CI/CD pipelines with AWS CodePipeline. This requires elevated permissions that should only be used for deployment.

AWS recommends that you create general purpose IAM resources, and if you plan to use App2Container to deploy your containers or create pipelines, that you create separate IAM resources for deployment which has elevated rights.

For simplicity in this guide, you will create a user with Administrator rights so it can deploy a containerized application using the AWS services for deployment that are supported by App2Container.

To create the user:

1. Navigate to the IAM service in the AWS Management Console. In the left pane, choose **Users > Add user**.
2. Select the **Programmatic access** type.
3. Choose **Next: Permissions**.



Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* app2container

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* [✓] **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

[ ] **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

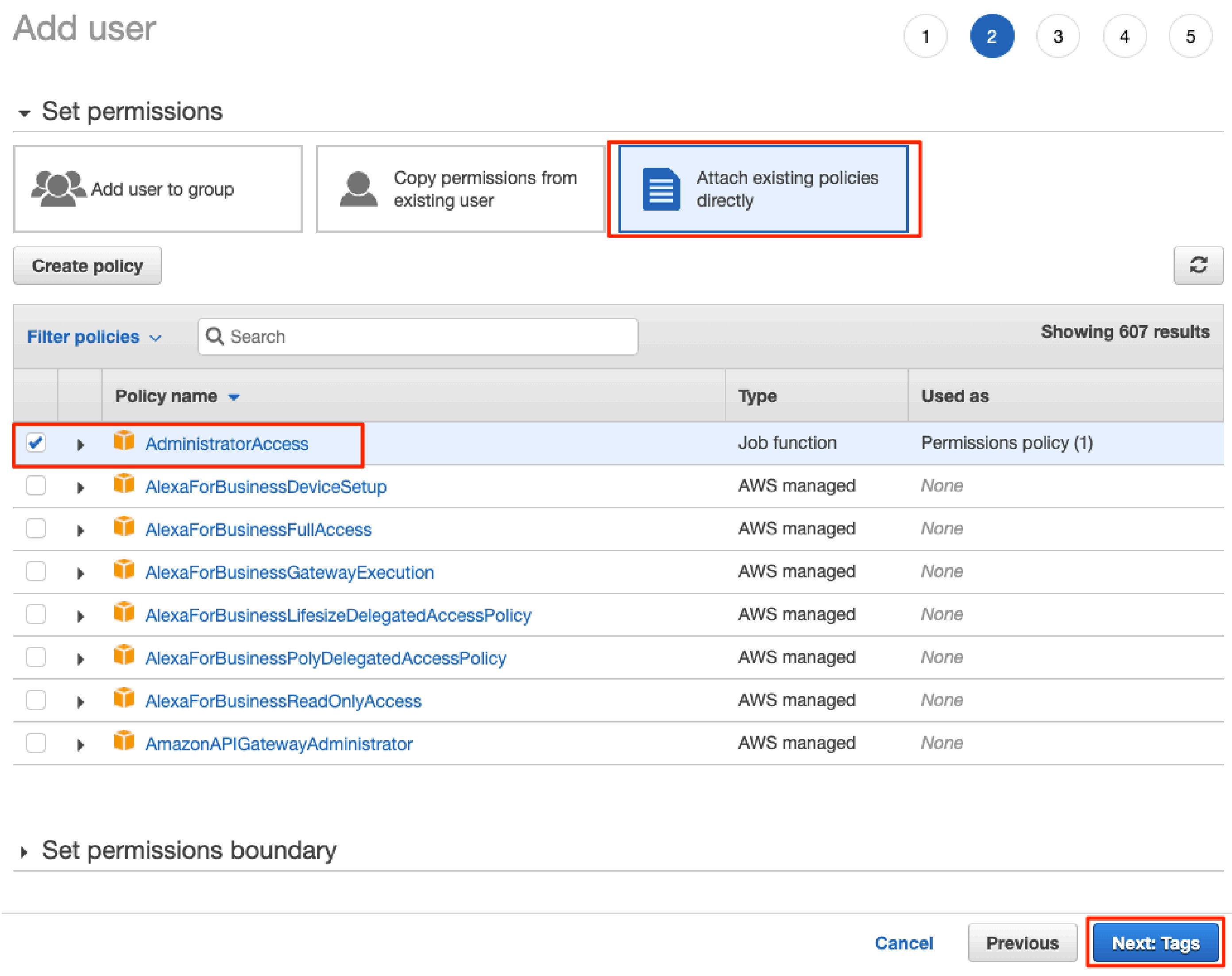
* Required

***Add user** dialog box*

4. Set permissions for the App2container user by choosing **Attach existing policies directory**.

5. Select **AdministratorAccess**, and choose **Next: Tags**.

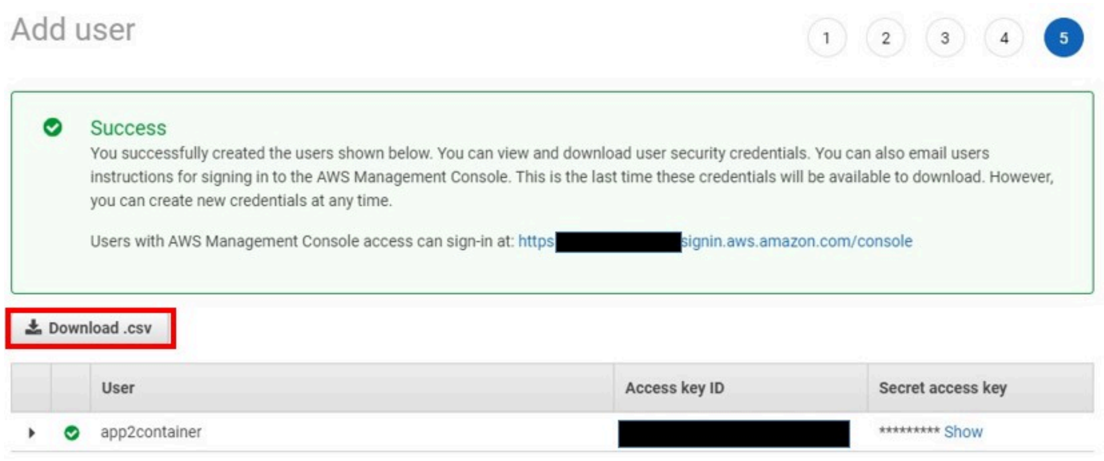
> **Note**
>
> **AdministratorAccess** should be used only for demonstration purposes. Review the official documentation for real use cases.



	Policy name	Type	Used as
✓	AdministratorAccess	Job function	Permissions policy (1)
	AlexaForBusinessDeviceSetup	AWS managed	*None*
	AlexaForBusinessFullAccess	AWS managed	*None*
	AlexaForBusinessGatewayExecution	AWS managed	*None*
	AlexaForBusinessLifesizeDelegatedAccessPolicy	AWS managed	*None*
	AlexaForBusinessPolyDelegatedAccessPolicy	AWS managed	*None*
	AlexaForBusinessReadOnlyAccess	AWS managed	*None*
	AmazonAPIGatewayAdministrator	AWS managed	*None*

*Select **AdministratorAccess**, and choose **Next: Tags***

6. Review and create your user. On the following screen, download the **access key ID** and **secret access key** to your local machine.



Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: https[redacted]signin.aws.amazon.com/console

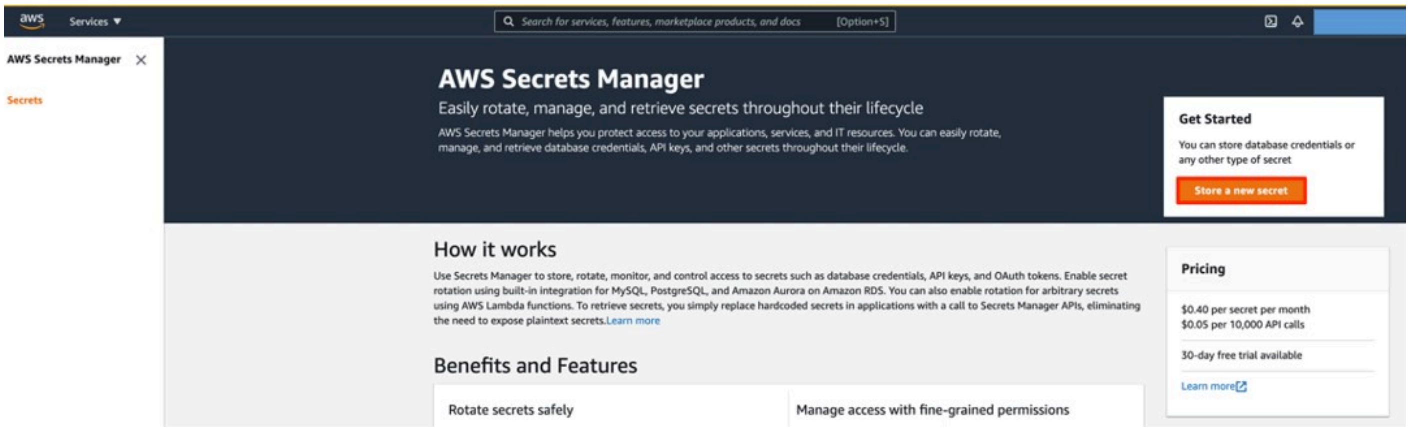
		User	Access key ID	Secret access key
▸	✓	app2container	[redacted]	********* Show

*Download the **access key ID** and **secret access key** to your local machine*

App2Container uses AWS Secrets Manager to manage the credentials for connecting your worker machine to application servers to run remote commands. Secrets Manager encrypts your secrets for storage, and provides an [Amazon Resource Name](https://) (ARN) for you to access the secret. When you run the remote configure command, you provide the secret ARN for App2Container to use to connect to your target server when running the remote command.

To store a new secret:

1. Navigate to AWS Secrets Manager in the AWS Management Console, and choose **Store a new secret**.


aws
Services ▼
Search for services, features, marketplace products, and docs [Option+S]
AWS Secrets Manager
Secrets
Store a new secret
Learn more

AWS Secrets Manager

Easily rotate, manage, and retrieve secrets throughout their lifecycle

AWS Secrets Manager helps you protect access to your applications, services, and IT resources. You can easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle.

Get Started

You can store database credentials or any other type of secret

How it works

Use Secrets Manager to store, rotate, monitor, and control access to secrets such as database credentials, API keys, and OAuth tokens. Enable secret rotation using built-in integration for MySQL, PostgreSQL, and Amazon Aurora on Amazon RDS. You can also enable rotation for arbitrary secrets using AWS Lambda functions. To retrieve secrets, you simply replace hardcoded secrets in applications with a call to Secrets Manager APIs, eliminating the need to expose plaintext secrets.Learn more

Pricing

$0.40 per secret per month
$0.05 per 10,000 API calls

30-day free trial available

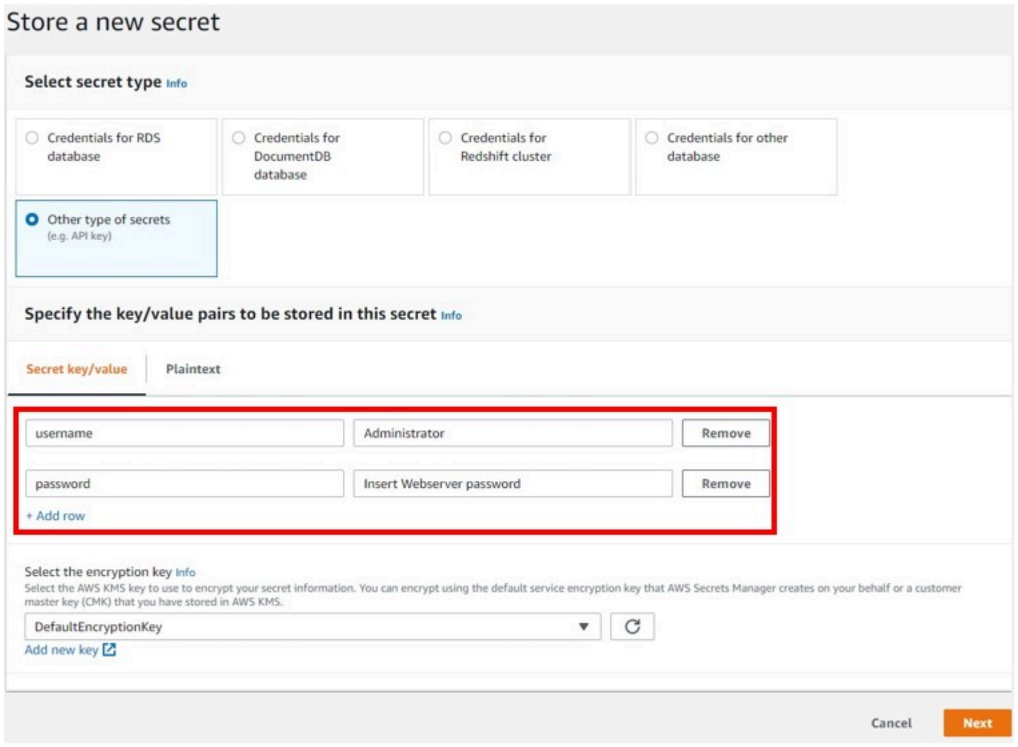
Benefits and Features

Rotate secrets safely

Manage access with fine-grained permissions

*Choose **Store a new secret***

2. Choose **Other type of secrets** and add the following parameters. Choose **Next**.


Store a new secret

Select secret type Info

- ○ Credentials for RDS database
- ○ Credentials for DocumentDB database
- ○ Credentials for Redshift cluster
- ○ Credentials for other database
- ● Other type of secrets (e.g. API key)

Specify the key/value pairs to be stored in this secret Info

Secret key/value | Plaintext

username	Administrator
password	Insert Webserver password

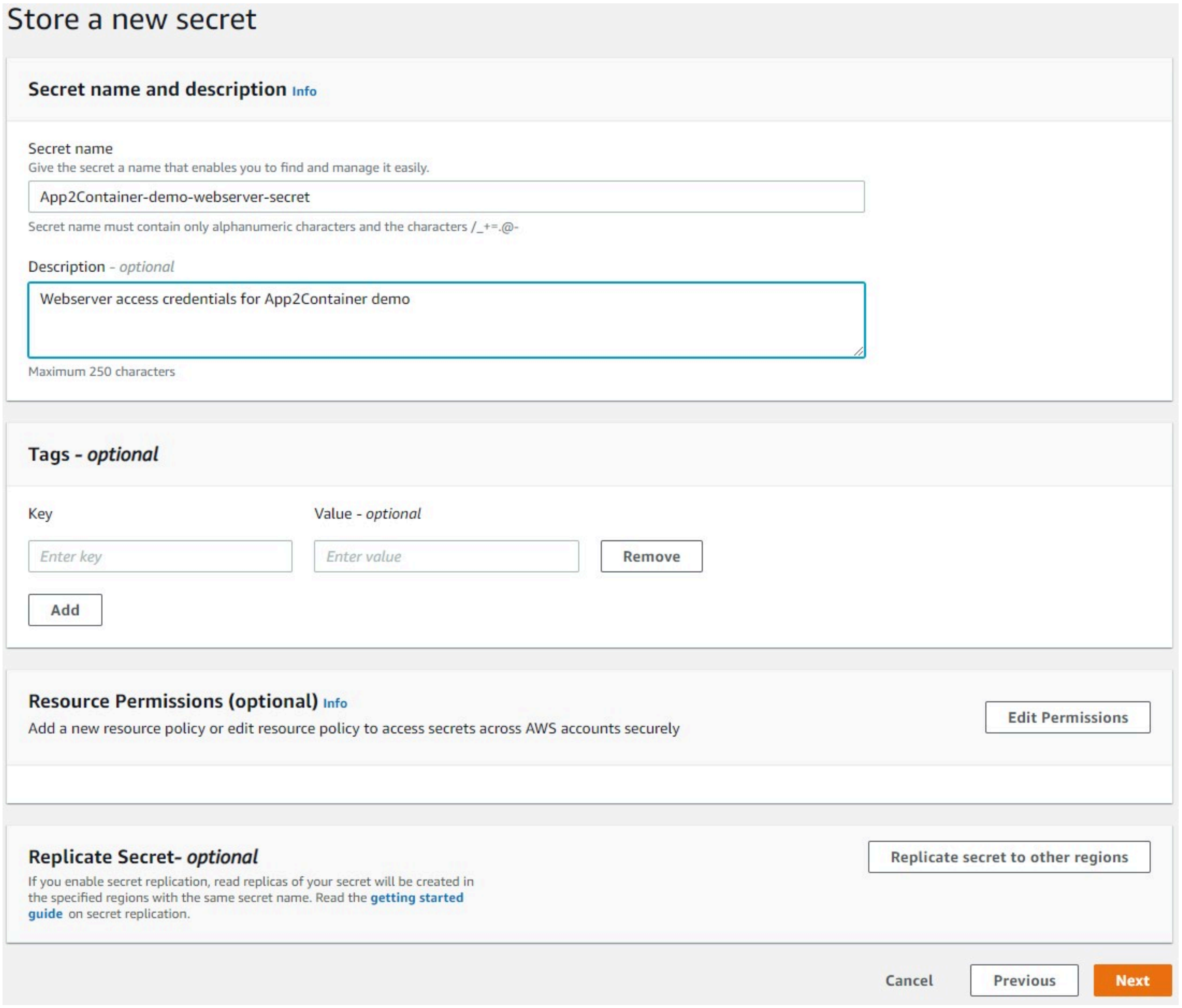
Select the encryption key Info

Select the AWS KMS key to use to encrypt your secret information. You can encrypt using the default service encryption key that AWS Secrets Manager creates on your behalf or a customer master key (CMK) that you have stored in AWS KMS.

DefaultEncryptionKey

*Add the parameters and choose **Next***

3. In **Name and description**, enter a secret name and description. Choose **Next**.


Store a new secret

Secret name and description Info

Secret name
Give the secret a name that enables you to find and manage it easily.
App2Container-demo-webserver-secret
Secret name must contain only alphanumeric characters and the characters /_+=.@-

Description - optional
Webserver access credentials for App2Container demo
Maximum 250 characters

Tags - optional

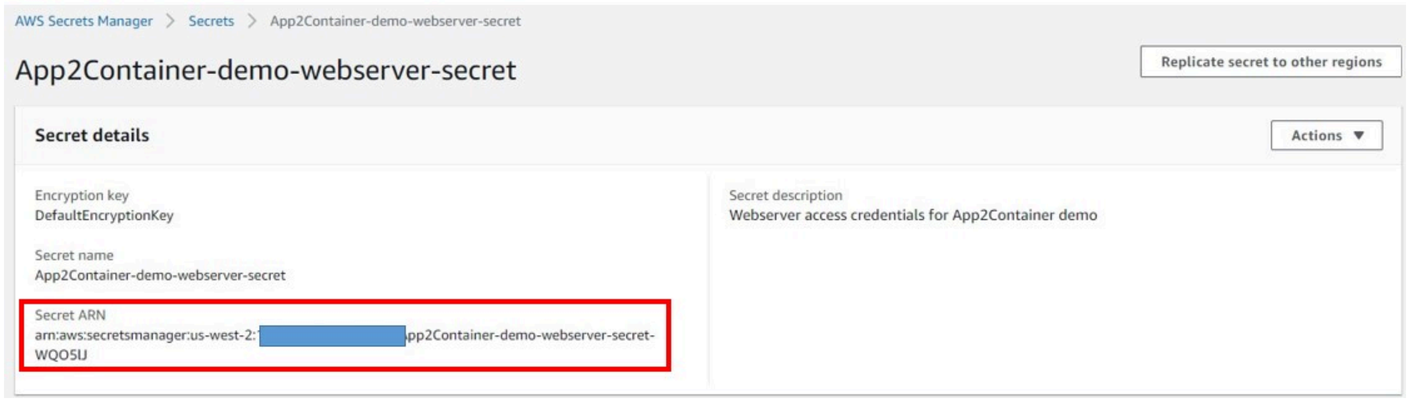
Key: Enter key
Value - optional: Enter value

Resource Permissions (optional) Info
Add a new resource policy or edit resource policy to access secrets across AWS accounts securely

Replicate Secret- optional
If you enable secret replication, read replicas of your secret will be created in the specified regions with the same secret name. Read the getting started guide on secret replication.

*Enter a secret name and description, and choose **Next***

4. On the next screen, leave the defaults in place, and choose **Next**.
5. After you store the password, choose it from the **Secrets** list. This will take you to a screen with the secret details. Copy the secret ARN to your local machine, because you will need this later.


AWS Secrets Manager > Secrets > App2Container-demo-webserver-secret
Replicate secret to other regions
Actions

App2Container-demo-webserver-secret

Secret details

Encryption key
DefaultEncryptionKey

Secret name
App2Container-demo-webserver-secret

Secret ARN
arn:aws:secretsmanager:us-west-2:[redacted]pp2Container-demo-webserver-secret-WQO5IJ

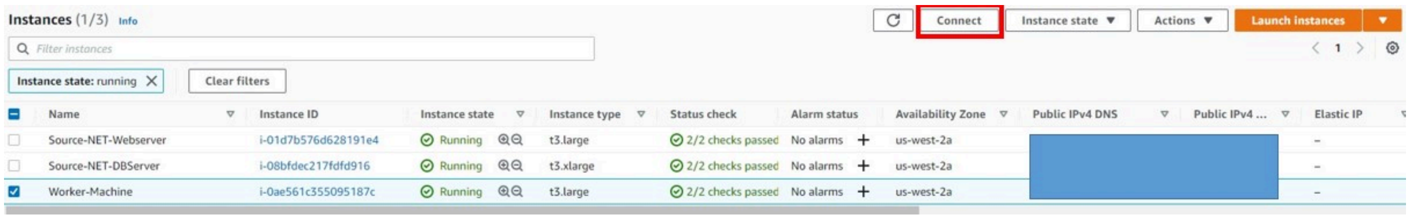
Secret description
Webserver access credentials for App2Container demo

Retrieve the secret ARN from the Secrets detail page

Now that you have your IAM role and secret created, log in to your worker machine and configure the AWS CLI with these newly created access objects.

To configure the AWS CLI:

1. Go to the EC2 service in the AWS Management Console, choose your worker machine instance, and choose **Connect** in the upper right of the screen.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Source-NET-Webserver	i-01d7b576d628191e4	Running	t3.large	2/2 checks passed	No alarms	us-west-2a	[redacted]	[redacted]	–
Source-NET-DBServer	i-08bfdec217fdfd916	Running	t3.xlarge	2/2 checks passed	No alarms	us-west-2a	[redacted]	[redacted]	–
Worker-Machine	i-0ae561c355095187c	Running	t3.large	2/2 checks passed	No alarms	us-west-2a	[redacted]	[redacted]	–

*Choose your worker machine instance, and choose **Connect** in the upper right of the screen*

2. Follow the same steps as detailed earlier in the *Connect to deployment* section to get the password for the worker machine. Copy that password to your local machine and use it to connect to the worker machine using Remote Desktop Connection.

3. When you are connected to the worker machine, open PowerShell and run the following:

```
aws configure

AWS Access Key ID [None]: <<add AWS access key from previous steps>>
AWS Secret Access Key [None]: <<add AWS secret access key from previous steps>>
Default region name [None]: us-west-2
Default output format [None]: [blank]
```

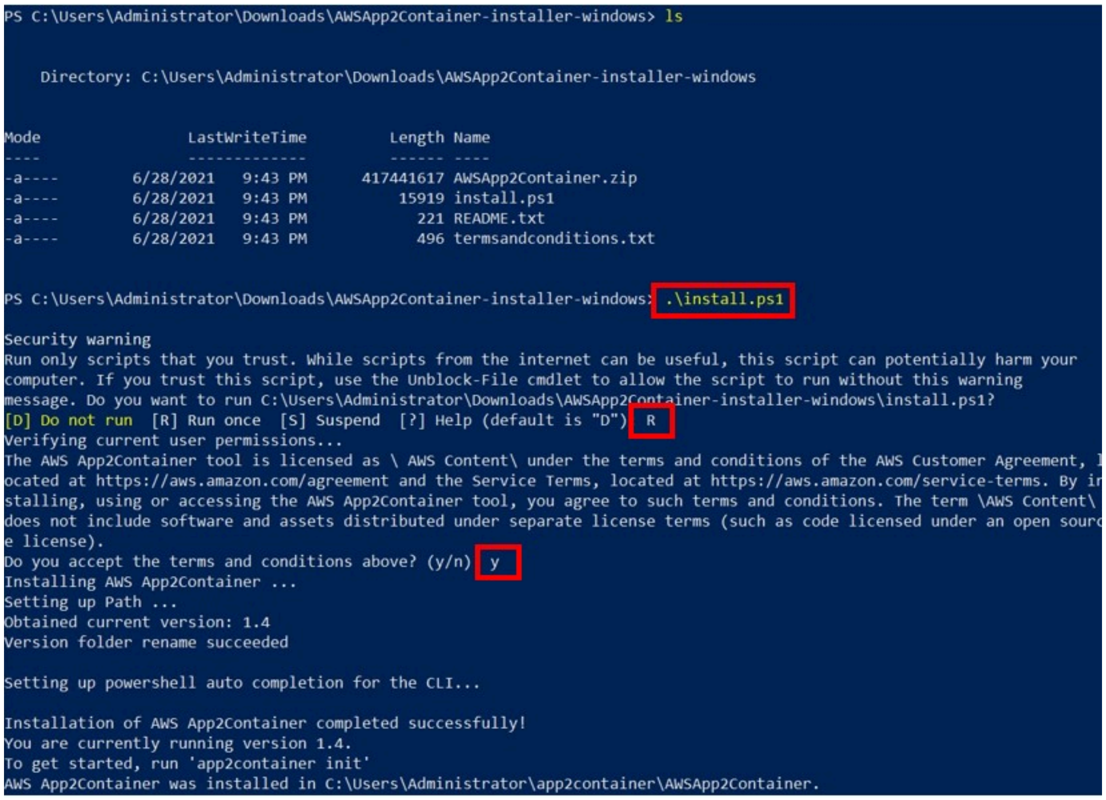
With this step, you have set up your environment prerequisites, and are ready for App2Container installation on your environment. In the next section, you will install App2Container on your worker machine and set it up to start your containerization process.

Install and initialize App2Container

Now that you have set up your environment for replatforming, you will install and initialize App2Container on your worker machine. App2Container can be run locally on the same machine as the application, or it can be run remotely from a separate machine. This walkthrough uses the latter approach, using App2Container's remote functionality.

To install and initialize App2Container on your worker machine:

1. On the worker machine, open a web browser and download the AWSApp2Container Windows installer to your server.
2. The installer file is saved to the **Downloads** folder. Navigate to the **Downloads** folder and extract the `AWSApp2Container-installer-windows.zip` file.
3. Run the `install.ps1` PowerShell script and enter **R** when prompted by the command output. Then press **y** to accept the terms and conditions.

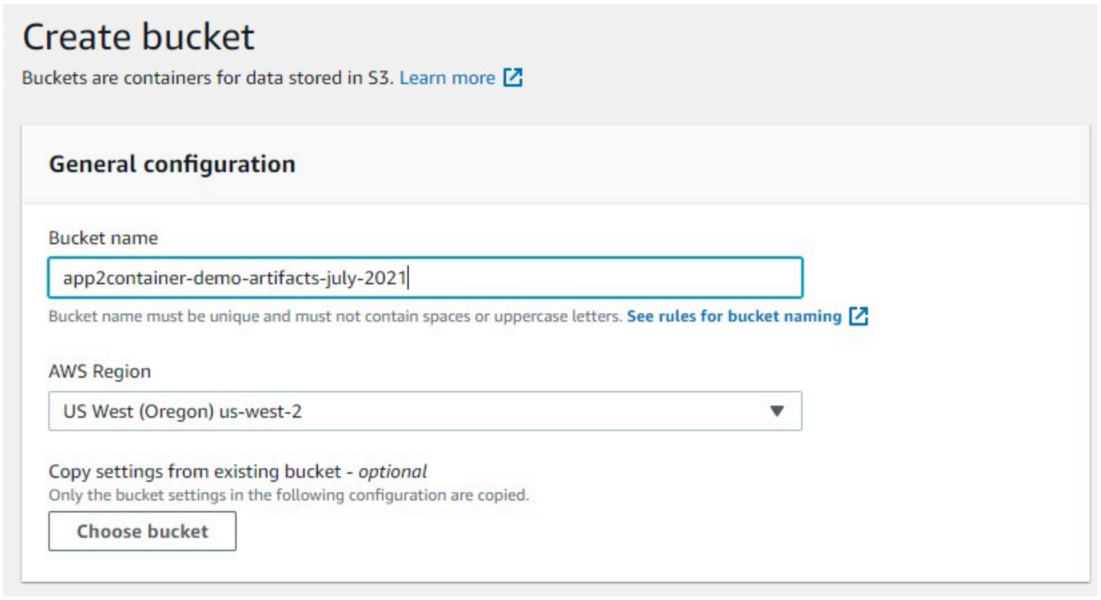


```
PS C:\Users\Administrator\Downloads\AWSApp2Container-installer-windows> ls


    Directory: C:\Users\Administrator\Downloads\AWSApp2Container-installer-windows


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        6/28/2021   9:43 PM      417441617 AWSApp2Container.zip
-a----        6/28/2021   9:43 PM          15919 install.ps1
-a----        6/28/2021   9:43 PM            221 README.txt
-a----        6/28/2021   9:43 PM            496 termsandconditions.txt


PS C:\Users\Administrator\Downloads\AWSApp2Container-installer-windows> .\install.ps1

Security warning
Run only scripts that you trust. While scripts from the internet can be useful, this script can potentially harm your
computer. If you trust this script, use the Unblock-File cmdlet to allow the script to run without this warning
message. Do you want to run C:\Users\Administrator\Downloads\AWSApp2Container-installer-windows\install.ps1?
[D] Do not run  [R] Run once  [S] Suspend  [?] Help (default is "D"): R
Verifying current user permissions...
The AWS App2Container tool is licensed as \ AWS Content\ under the terms and conditions of the AWS Customer Agreement, l
ocated at https://aws.amazon.com/agreement and the Service Terms, located at https://aws.amazon.com/service-terms. By in
stalling, using or accessing the AWS App2Container tool, you agree to such terms and conditions. The term \AWS Content\
does not include software and assets distributed under separate license terms (such as code licensed under an open sour
e license).
Do you accept the terms and conditions above? (y/n): y
Installing AWS App2Container ...
Setting up Path ...
Obtained current version: 1.4
Version folder rename succeeded

Setting up powershell auto completion for the CLI...

Installation of AWS App2Container completed successfully!
You are currently running version 1.4.
To get started, run 'app2container init'
AWS App2Container was installed in C:\Users\Administrator\app2container\AWSApp2Container.
```

`install.ps1` PowerShell script

Now you will perform the one-time initialization command for App2Container. This interactive command prompts for the information required to set up the App2Container environment.

4. Go to Amazon S3 in the AWS Management Console and create an S3 bucket where App2Container will store artifacts during the containerization process. Enter a unique name for your bucket.



Create bucket

Buckets are containers for data stored in S3. Learn more

General configuration

Bucket name

app2container-demo-artifacts-july-2021

Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming

AWS Region

US West (Oregon) us-west-2

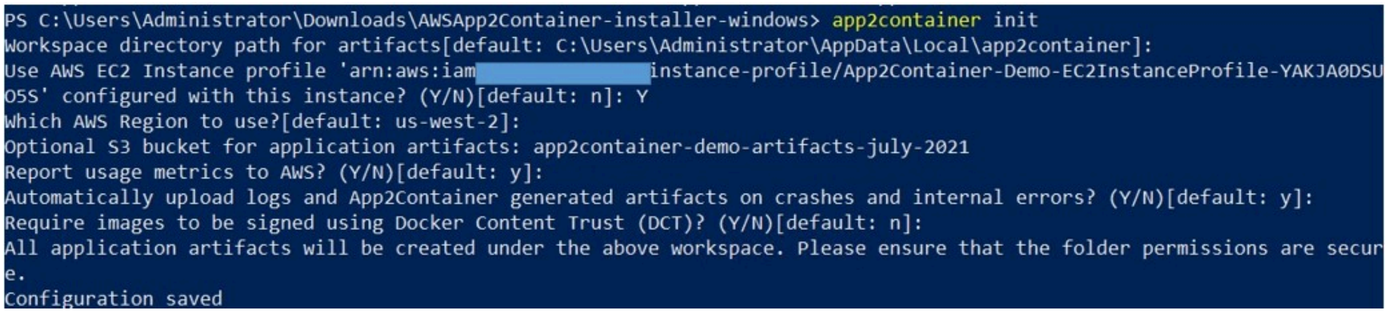
Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Create bucket *dialog box*

5. On your worker machine, run the following command.

```
app2container init
```



```
PS C:\Users\Administrator\Downloads\AWSApp2Container-installer-windows> app2container init
Workspace directory path for artifacts[default: C:\Users\Administrator\AppData\Local\app2container]:
Use AWS EC2 Instance profile 'arn:aws:iam[redacted]instance-profile/App2Container-Demo-EC2InstanceProfile-YAKJA0DSU
O5S' configured with this instance? (Y/N)[default: n]: Y
Which AWS Region to use?[default: us-west-2]:
Optional S3 bucket for application artifacts: app2container-demo-artifacts-july-2021
Report usage metrics to AWS? (Y/N)[default: y]:
Automatically upload logs and App2Container generated artifacts on crashes and internal errors? (Y/N)[default: y]:
Require images to be signed using Docker Content Trust (DCT)? (Y/N)[default: n]:
All application artifacts will be created under the above workspace. Please ensure that the folder permissions are secur
e.
Configuration saved
```

Run the `app2container init` command

Table 5 — App2Container PowerShell initialization parameter description

Parameter	Value
Workspace directory path	Leave default (`C:\Users\Administrator\AppData\Local\app2container`)
AWS profile	Y (Contains information needed to run App2Container)
S3 bucket	Enter the S3 bucket name that you created in the previous step (such as `app2container-demo-artifacts-july-2021`)
AWS Region	`us-west-2` (default)
Permission to collect metrics	Leave default (allow App2Container to collect information about the host operating system, app type, and the commands run)
Enforce signed images	Leave default (optionally require that images are signed using Docker Content Trust)

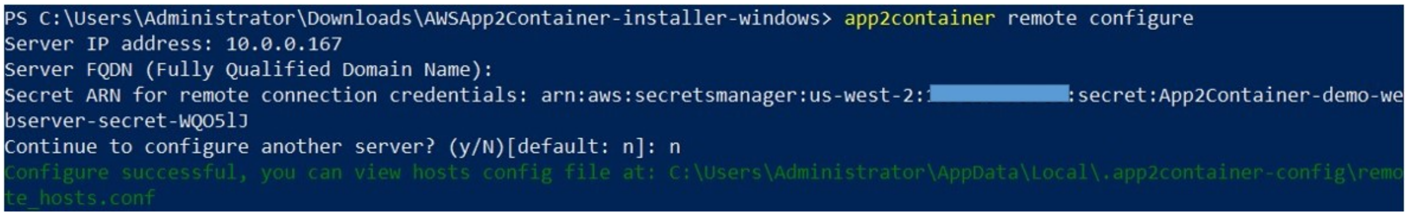
6. Now you need to give App2Container the information it needs to access your web server instance. You do this by running the following command and providing the following information in the command prompts.

```
app2container remote configure
```

Table 6 — App2Container web server access parameters

Parameter	Value
Server IP address	Web server Private IPv4 address (found at the **EC2** > **Source-NET-Webserver** > **Details** > **Private IPv4 Address**)
Server FQDN	Leave blank

Parameter	Value
Secret ARN	ARN for the secret you created in AWS Secret Manager
Continue to another server?	n



```
PS C:\Users\Administrator\Downloads\AWSApp2Container-installer-windows> app2container remote configure
Server IP address: 10.0.0.167
Server FQDN (Fully Qualified Domain Name):
Secret ARN for remote connection credentials: arn:aws:secretsmanager:us-west-2:[redacted]:secret:App2Container-demo-we
bserver-secret-WQO5lJ
Continue to configure another server? (y/N)[default: n]: n
Configure successful, you can view hosts config file at: C:\Users\Administrator\AppData\Local\.app2container-config\remo
te_hosts.conf
```

Configuring App2Container for remote access to IIS web server

This concludes the prerequisite steps to run App2Container on your worker machine. In the next section, you will use App2Container to discover, analyze, and containerize the MvcMusicStore application that is running on the web server instance without directly touching the application server.

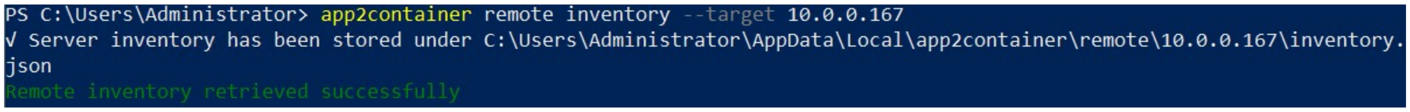
Containerization

Because you are using the worker machine for all of your App2container process, you will use a remote process. After you run the remote commands from your worker machine, it will connect with your application web server and run the commands on it.

The following steps will allow you to containerize .NET applications running on a remote server using App2Container:

1. Run the `app2container remote inventory` command (as follows) for App2Container to gather the .NET applications that are running on your web server.

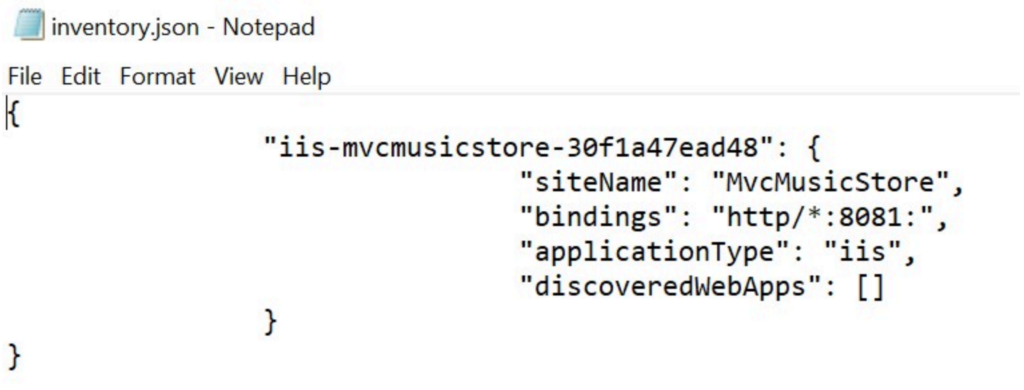
```
app2container remote inventory --target &lt;webserver-private-ip>
```



```
PS C:\Users\Administrator> app2container remote inventory --target 10.0.0.167
√ Server inventory has been stored under C:\Users\Administrator\AppData\Local\app2container\remote\10.0.0.167\inventory.
json
Remote inventory retrieved successfully
```

App2Container output for retrieving the inventory of .NET applications from a server

2. Confirm that your application is listed in the `inventory.json` file at the location specified in the tool's output.



inventory.json - Notepad

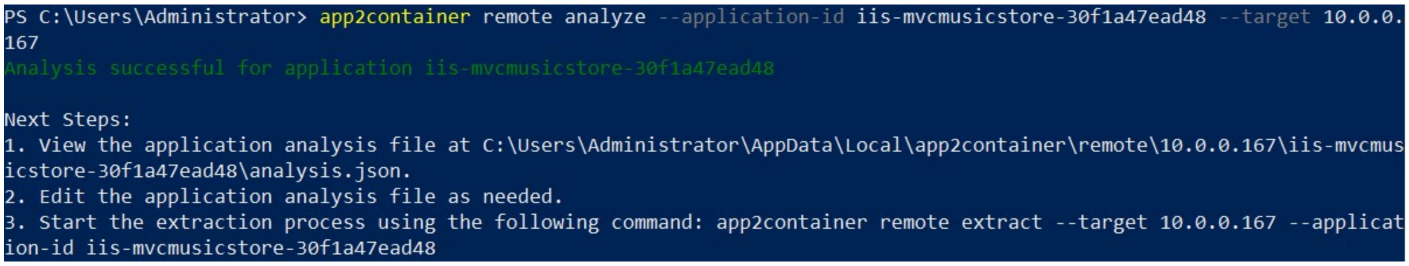
File Edit Format View Help

```
{
                "iis-mvcmusicstore-30f1a47ead48": {
                                "siteName": "MvcMusicStore",
                                "bindings": "http/*:8081:",
                                "applicationType": "iis",
                                "discoveredWebApps": []
                }
}
```

Confirm that your application is listed in the `inventory.json` file

3. Locate the application ID for the application in the `inventory.json` file and then run the following command, replacing `net-app-id` with the application ID that you located in the `inventory.json`.

```
 app2container remote analyze --application-id <net-app-id> --target <webserver-
private-ip>
```



```
PS C:\Users\Administrator> app2container remote analyze --application-id iis-mvcmusicstore-30f1a47ead48 --target 10.0.0.
167
Analysis successful for application iis-mvcmusicstore-30f1a47ead48

Next Steps:
1. View the application analysis file at C:\Users\Administrator\AppData\Local\app2container\remote\10.0.0.167\iis-mvcmus
icstore-30f1a47ead48\analysis.json.
2. Edit the application analysis file as needed.
3. Start the extraction process using the following command: app2container remote extract --target 10.0.0.167 --applicat
ion-id iis-mvcmusicstore-30f1a47ead48
```

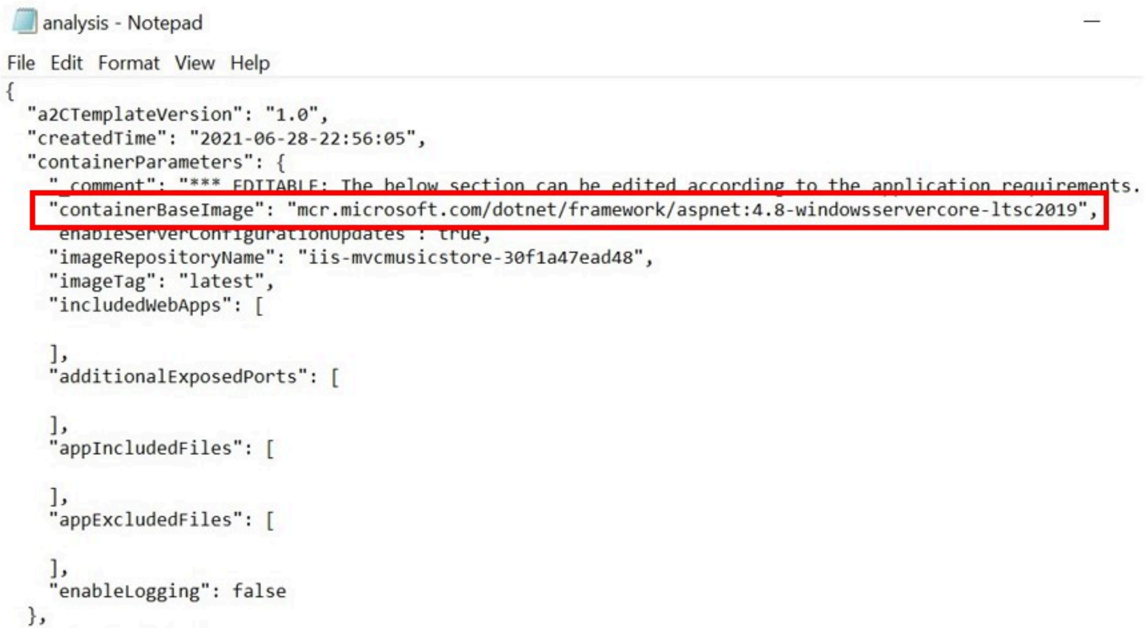
App2Container successfully analyzes a .NET application for containerization

4. Open the `analysis.json` file at the location specified in the tool's output.

   This file has two sections, EDITABLE and NON-EDITABLE. The EDITABLE section includes container parameters that specify settings such as `containerBaseImage` or image tag. These parameters will be used in the subsequent containerization process.

   The `containerBaseImage` is not populated in the file. This is because App2Container will automatically configure the image during the next containerization step using the worker machine version to determine the `containerBaseImage` to use. However, to provide visibility into the behavior, populate that field in the `analysis.json` file with `mcr.microsoft.com/dotnet/framework/aspnet:4.8-windowsservercore-ltsc2019`, as shown in the following figure, and save the file.

> **Note**
>
> This guide recommends that you use the SAC version for container-based applications because it has the most up-to-date improvements for Windows containers. However, App2Container requires the `containerBaseImage` to match the worker machine version and SAC version 2004 does not have a desktop experience. We chose LTSC version 2019 to provide a more effective walkthrough. If you use SAC version 2004, deploy a worker machine of that version and replace `containerBaseImage` with the `4.8-windowsservercore-2004` tag.



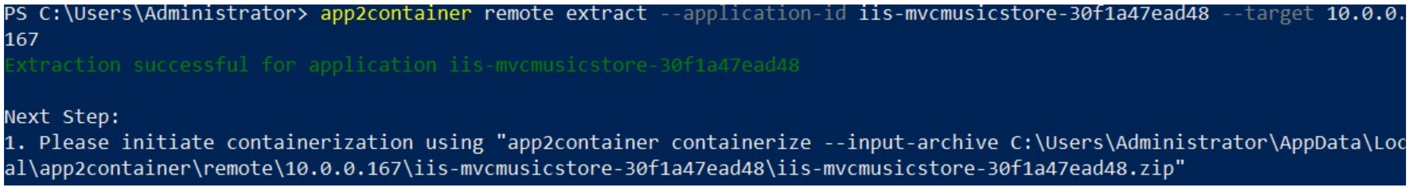
```
{
  "a2CTemplateVersion": "1.0",
  "createdTime": "2021-06-28-22:56:05",
  "containerParameters": {
    "_comment": "*** EDITABLE: The below section can be edited according to the application requirements.
    "containerBaseImage": "mcr.microsoft.com/dotnet/framework/aspnet:4.8-windowsservercore-ltsc2019",
    "enableServerConfigurationUpdates": true,
    "imageRepositoryName": "iis-mvcmusicstore-30f1a47ead48",
    "imageTag": "latest",
    "includedWebApps": [

    ],
    "additionalExposedPorts": [

    ],
    "appIncludedFiles": [

    ],
    "appExcludedFiles": [

    ],
    "enableLogging": false
  },
```

Populate the field in the `analysis.json` file with `mcr.microsoft.com/dotnet/framework/aspnet:4.8-windowsservercore-ltsc2019`

The NON-EDITABLE section of `analysis.json` includes application-level information that App2Container uses during the containerization step such as OS data, ports in use, dependencies, software libraries, and so on.

5. Now you will transform your application with App2Container by extracting the artifacts from the worker machine which will be used to produce the container image. The extraction process takes a few minutes and App2Container will create a ZIP file that includes all of the artifacts. As a suggested next step from the `app2container analyze` command output, run the `app2container remote extract` command to retrieve all application artifacts to the worker machine. Update the `application-id` and target IP address based on your environment.

   ```
    app2container remote extract --application-id <net-app-id> --target <webserver-
   private-ip>
   ```

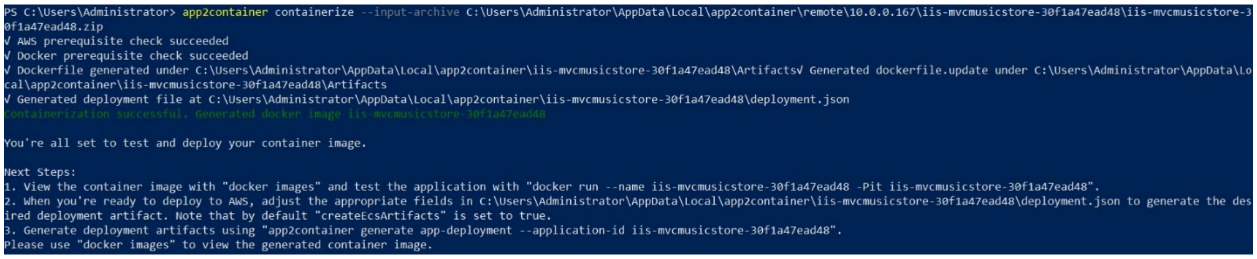
   ```
   PS C:\Users\Administrator> app2container remote extract --application-id iis-mvcmusicstore-30f1a47ead48 --target 10.0.0.
   167
   Extraction successful for application iis-mvcmusicstore-30f1a47ead48

   Next Step:
   1. Please initiate containerization using "app2container containerize --input-archive C:\Users\Administrator\AppData\Loc
   al\app2container\remote\10.0.0.167\iis-mvcmusicstore-30f1a47ead48\iis-mvcmusicstore-30f1a47ead48.zip"
   ```

   App2Container successfully extracting the artifacts required for containerization

6. Now you will run the `app2container containerize` command to create the container image based on the extracted files. This process will take several minutes, because the worker machine needs to download the container image .NET framework base layers, extract, and build the container image.

   ```
    app2container containerize --input-archive C:\Users\Administrator\AppData\Local
   \app2container\remote\<webserver-private-ip>\<application-id>\<application-id>
   ```

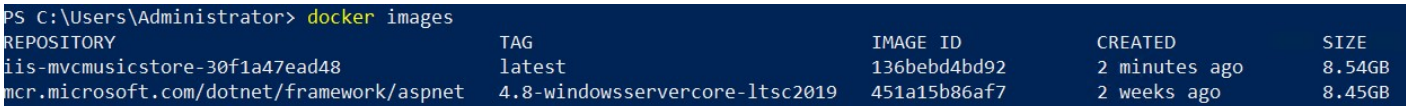
   After the process completes, you will see the following output:

   ```
   PS C:\Users\Administrator> app2container containerize --input-archive C:\Users\Administrator\AppData\Local\app2container\remote\10.0.0.167\iis-mvcmusicstore-30f1a47ead48\iis-mvcmusicstore-3
   0f1a47ead48.zip
   √ AWS prerequisite check succeeded
   √ Docker prerequisite check succeeded
   √ Dockerfile generated under C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47ead48\Artifacts√ Generated dockerfile.update under C:\Users\Administrator\AppData\Lo
   cal\app2container\iis-mvcmusicstore-30f1a47ead48\Artifacts
   √ Generated deployment file at C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47ead48\deployment.json
   Containerization successful. Generated docker image iis-mvcmusicstore-30f1a47ead48

   You're all set to test and deploy your container image.

   Next Steps:
   1. View the container image with "docker images" and test the application with "docker run --name iis-mvcmusicstore-30f1a47ead48 -Pit iis-mvcmusicstore-30f1a47ead48".
   2. When you're ready to deploy to AWS, adjust the appropriate fields in C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47ead48\deployment.json to generate the des
   ired deployment artifact. Note that by default "createEcsArtifacts" is set to true.
   3. Generate deployment artifacts using "app2container generate app-deployment --application-id iis-mvcmusicstore-30f1a47ead48".
   Please use "docker images" to view the generated container image.
   ```

   App2Container successfully creates a container image of the .NET application

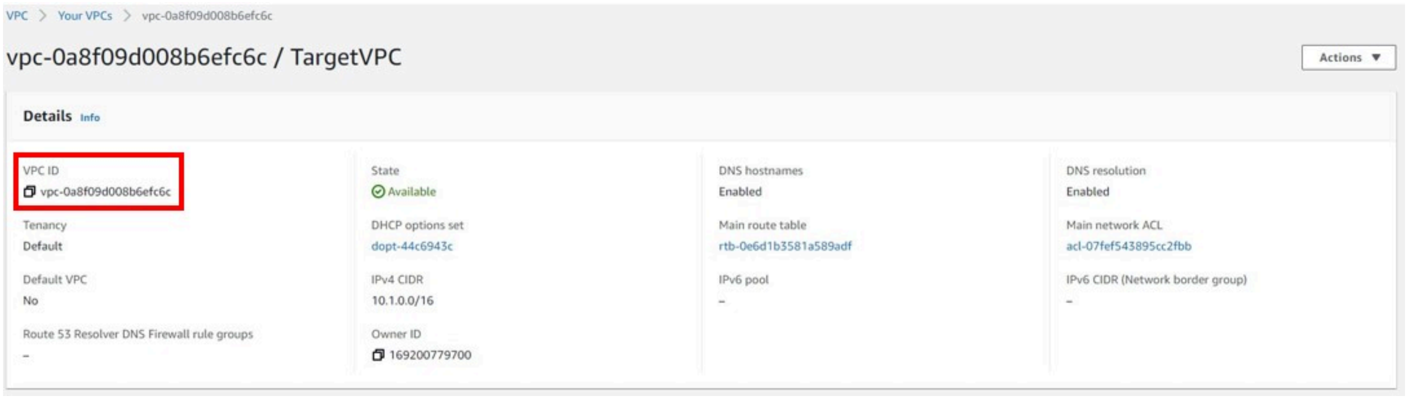
7. Run docker images to see the container image that App2Container created for you using the `windowsservercore-ltsc-2019` and `latest` image tag.

   ```
   PS C:\Users\Administrator> docker images
   REPOSITORY                               TAG                             IMAGE ID       CREATED          SIZE
   iis-mvcmusicstore-30f1a47ead48           latest                          136bebd4bd92   2 minutes ago    8.54GB
   mcr.microsoft.com/dotnet/framework/aspnet   4.8-windowsservercore-ltsc2019   451a15b86af7   2 weeks ago      8.45GB
   ```

List of container images available on the local machine

8. Running the app2container containerize command creates a `deployment.json` file inside the application folder as noted in the output of the command. This file includes the AWS deployment configurations and you can update this file to fit your desired target environment such as changing the VPC or defining the CPU/memory allocation. Open the `deployment.json` file and ensure that `createEcsArtifacts` is set to true and set the target `vpc-id` to deploy the application to the target environment that you deployed previously. You can find the target `vpc-id` in the VPC service in the AWS Management Console.



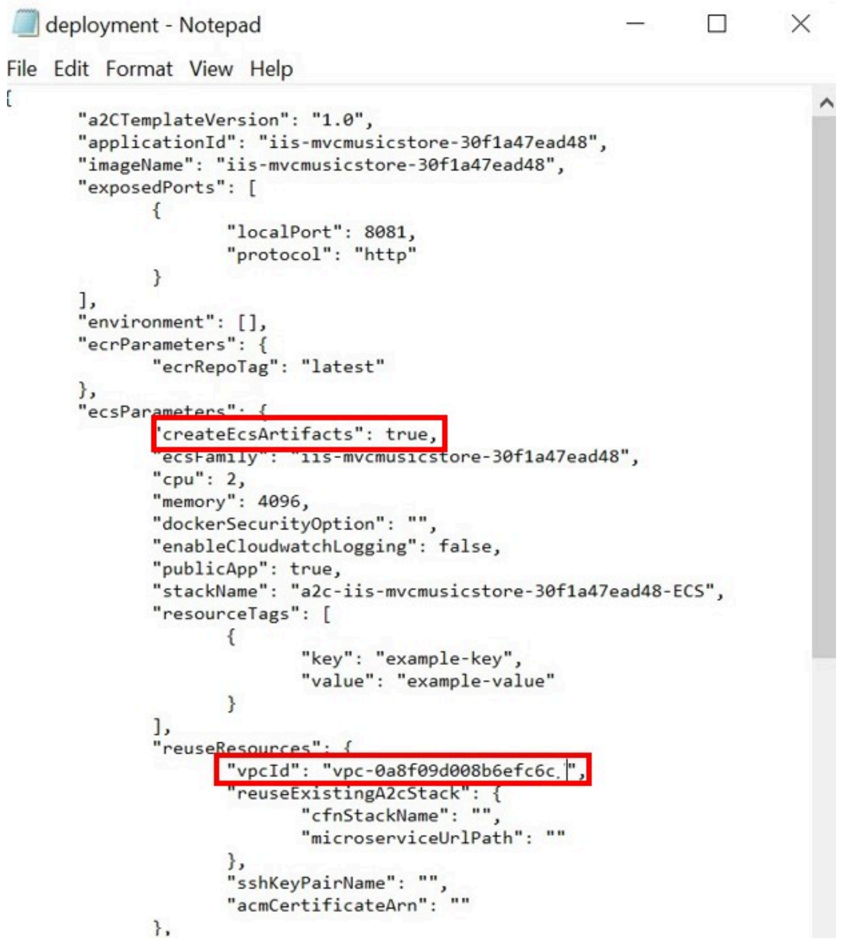
VPC > Your VPCs > vpc-0a8f09d008b6efc6c

vpc-0a8f09d008b6efc6c / TargetVPC

Details

VPC ID: vpc-0a8f09d008b6efc6c	State: Available	DNS hostnames: Enabled	DNS resolution: Enabled
Tenancy: Default	DHCP options set: dopt-44c6943c	Main route table: rtb-0e6d1b3581a589adf	Main network ACL: acl-07fef543895cc2fbb
Default VPC: No	IPv4 CIDR: 10.1.0.0/16	IPv6 pool: –	IPv6 CIDR (Network border group): –
Route 53 Resolver DNS Firewall rule groups: –	Owner ID: 169200779700		

Target `vpc-id`



```
{
        "a2CTemplateVersion": "1.0",
        "applicationId": "iis-mvcmusicstore-30f1a47ead48",
        "imageName": "iis-mvcmusicstore-30f1a47ead48",
        "exposedPorts": [
                {
                        "localPort": 8081,
                        "protocol": "http"
                }
        ],
        "environment": [],
        "ecrParameters": {
                "ecrRepoTag": "latest"
        },
        "ecsParameters": {
                "createEcsArtifacts": true,
                "ecsFamily": "iis-mvcmusicstore-30f1a47ead48",
                "cpu": 2,
                "memory": 4096,
                "dockerSecurityOption": "",
                "enableCloudwatchLogging": false,
                "publicApp": true,
                "stackName": "a2c-iis-mvcmusicstore-30f1a47ead48-ECS",
                "resourceTags": [
                        {
                                "key": "example-key",
                                "value": "example-value"
                        }
                ],
                "reuseResources": {
                        "vpcId": "vpc-0a8f09d008b6efc6c",
                        "reuseExistingA2cStack": {
                                "cfnStackName": "",
                                "microserviceUrlPath": ""
                        },
                        "sshKeyPairName": "",
                        "acmCertificateArn": ""
                },
```

Sample `deployment.json` file

You have successfully containerized your application, and are ready to deploy your newly containerized application to AWS. In the next section you will deploy your application to Amazon ECS.

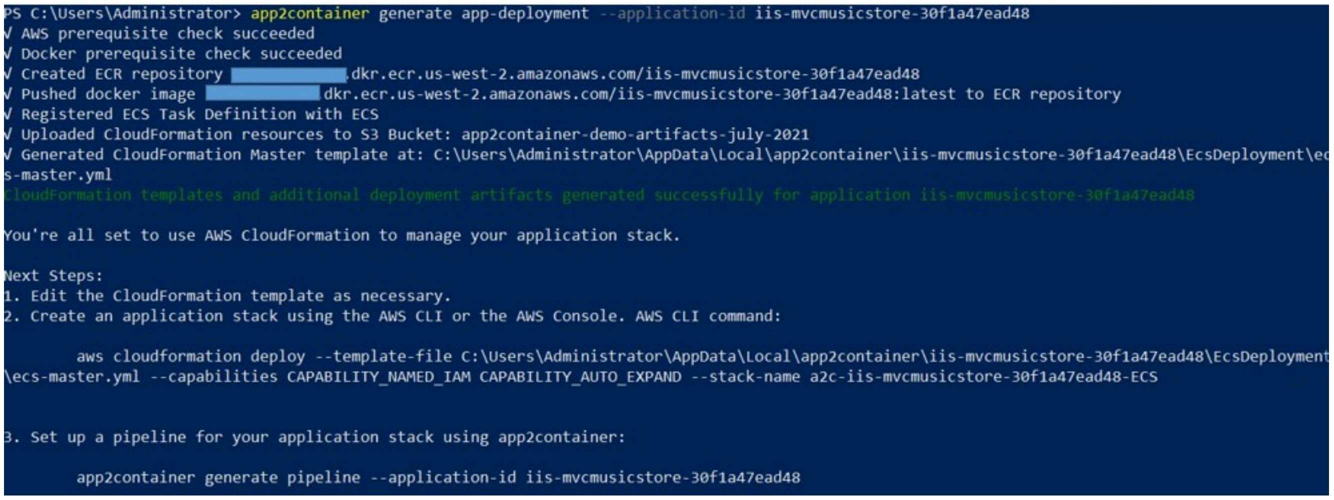
Deployment

You will use App2Container to generate the artifacts needed to deploy your application container in AWS. App2Container pre-fills key values in the artifacts based on your profile, the application analysis, and best practices. In this guide you will deploy your application to Amazon ECS; however, Amazon EKS is also a supported option.

1. Run `app2container generate app-deployment` to automatically create a CloudFormation template for deploying the MvcMusicStore application to Amazon ECS.

```
app2container generate app-deployment --application-id <net-app-id>
```

This process will take a few minutes to complete, and will result in the following output.



```
PS C:\Users\Administrator> app2container generate app-deployment --application-id iis-mvcmusicstore-30f1a47ead48
√ AWS prerequisite check succeeded
√ Docker prerequisite check succeeded
√ Created ECR repository [redacted].dkr.ecr.us-west-2.amazonaws.com/iis-mvcmusicstore-30f1a47ead48
√ Pushed docker image [redacted]dkr.ecr.us-west-2.amazonaws.com/iis-mvcmusicstore-30f1a47ead48:latest to ECR repository
√ Registered ECS Task Definition with ECS
√ Uploaded CloudFormation resources to S3 Bucket: app2container-demo-artifacts-july-2021
√ Generated CloudFormation Master template at: C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47ead48\EcsDeployment\ec
s-master.yml
CloudFormation templates and additional deployment artifacts generated successfully for application iis-mvcmusicstore-30f1a47ead48

You're all set to use AWS CloudFormation to manage your application stack.

Next Steps:
1. Edit the CloudFormation template as necessary.
2. Create an application stack using the AWS CLI or the AWS Console. AWS CLI command:

        aws cloudformation deploy --template-file C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47ead48\EcsDeployment
\ecs-master.yml --capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND --stack-name a2c-iis-mvcmusicstore-30f1a47ead48-ECS


3. Set up a pipeline for your application stack using app2container:

        app2container generate pipeline --application-id iis-mvcmusicstore-30f1a47ead48
```

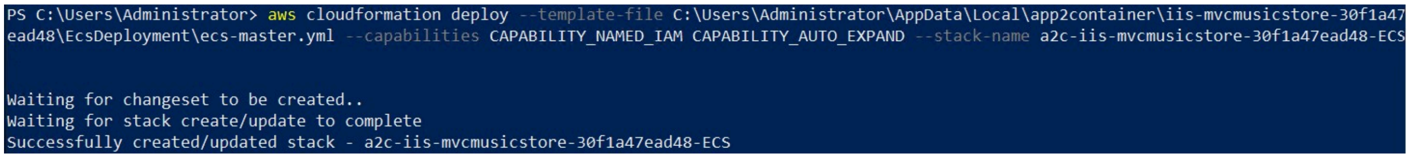
Successful generation of CloudFormation template for MvcMusic deployment

This process creates an Amazon ECR repository and pushes your app image to the registry. It also creates an ECS task definition and registers it with ECS, then uploads all CloudFormation resources to the selected S3 bucket. Lastly, it creates a CloudFormation template for the deployment and fills it with the generated configurations from the previous steps. You can verify the creation of these assets by viewing the services in the AWS Management Console and by confirming the `ecs-master.yml` file in the location specified in the output of the tool.

2. Launch the CloudFormation stack by running the suggested `aws cloudformation deploy` command to pin the tool's output. Replace the values in the following template with your actual value for your application's ID.

```
aws cloudformation deploy --template-file
C:\Users\Administrator\AppData\Local\app2container\<net-app-id>
\EcsDeployment\ecs-master.yml --capabilities CAPABILITY_NAMED_IAM
--stack-name a2c-<net-app-id>-ECS
```

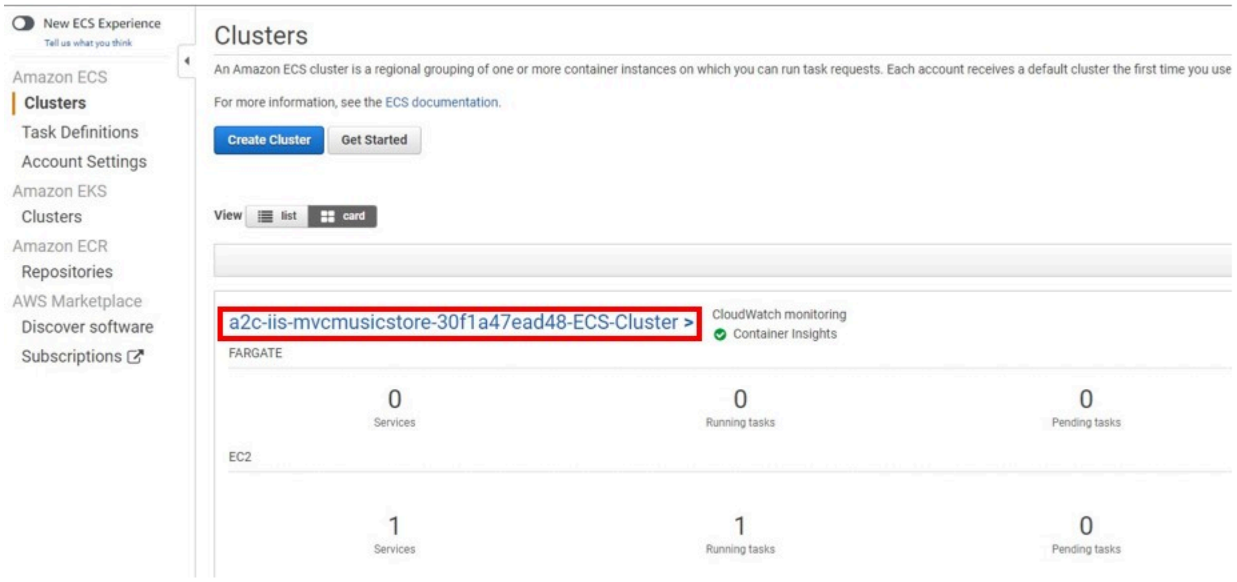
This process will take several minutes to complete while the infrastructure is deployed in AWS. You can track the progress in the CloudFormation service in the AWS Management Console. When the command completes, the result will be the following output.



```
PS C:\Users\Administrator> aws cloudformation deploy --template-file C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47
ead48\EcsDeployment\ecs-master.yml --capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND --stack-name a2c-iis-mvcmusicstore-30f1a47ead48-ECS


Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - a2c-iis-mvcmusicstore-30f1a47ead48-ECS
```

Successful deployment using CloudFormation template

3. After your stack is deployed, you can verify that your containerized application is successfully running by navigating to Amazon ECS in the EC2 console. In Amazon ECS, choose **Clusters** in the left pane, and choose the link for your newly created cluster.



Clusters

An Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests. Each account receives a default cluster the first time you use

For more information, see the ECS documentation.

a2c-iis-mvcmusicstore-30f1a47ead48-ECS-Cluster >

CloudWatch monitoring
Container Insights

FARGATE

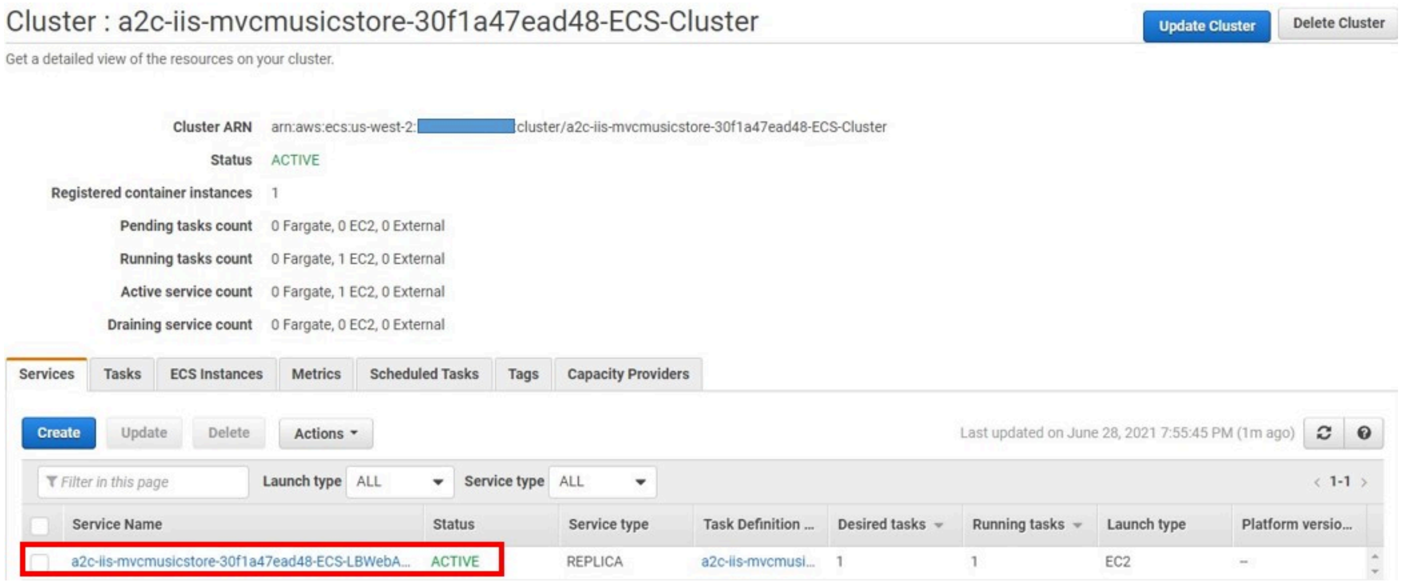
0 Services · 0 Running tasks · 0 Pending tasks

EC2

1 Services · 1 Running tasks · 0 Pending tasks

*Choose **Clusters** in the left pane and choose the link for your newly created cluster*

4. Select the link to Amazon ECS running on your ECS cluster.



Cluster : a2c-iis-mvcmusicstore-30f1a47ead48-ECS-Cluster

Get a detailed view of the resources on your cluster.

Cluster ARN	arn:aws:ecs:us-west-2:[redacted]cluster/a2c-iis-mvcmusicstore-30f1a47ead48-ECS-Cluster
Status	ACTIVE
Registered container instances	1
Pending tasks count	0 Fargate, 0 EC2, 0 External
Running tasks count	0 Fargate, 1 EC2, 0 External
Active service count	0 Fargate, 1 EC2, 0 External
Draining service count	0 Fargate, 0 EC2, 0 External

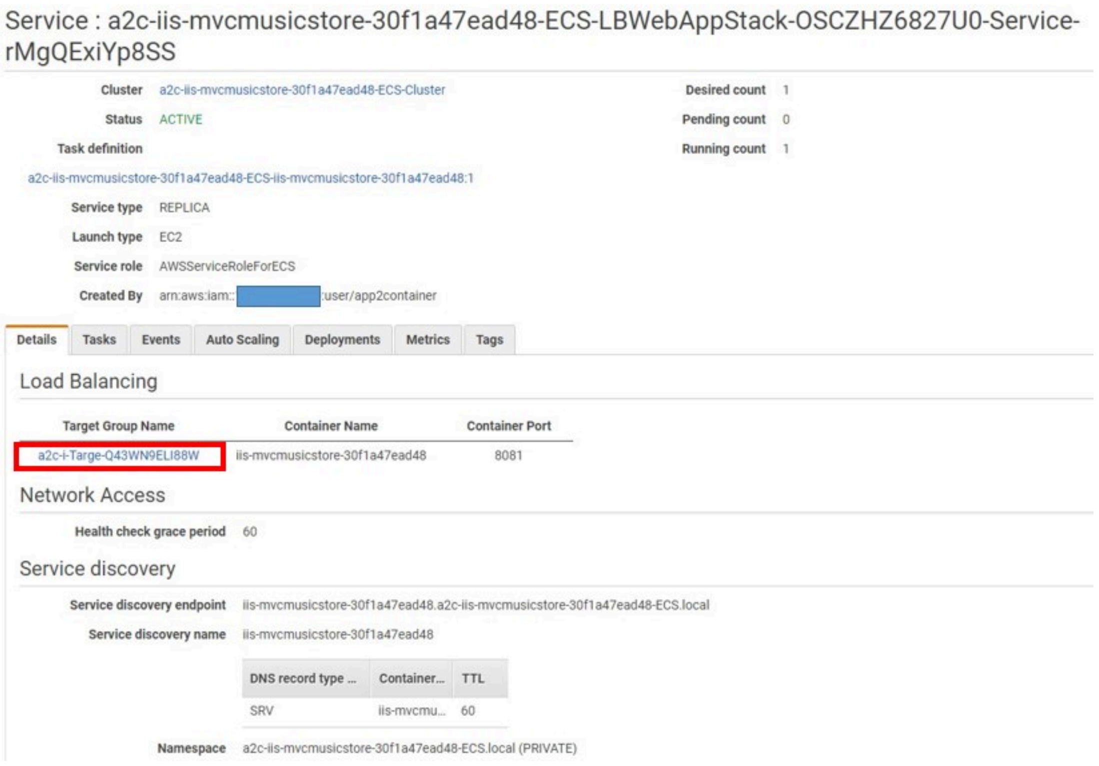
Last updated on June 28, 2021 7:55:45 PM (1m ago)

Launch type ALL · Service type ALL

Service Name	Status	Service type	Task Definition ...	Desired tasks	Running tasks	Launch type	Platform versio...
a2c-iis-mvcmusicstore-30f1a47ead48-ECS-LBWebA...	ACTIVE	REPLICA	a2c-iis-mvcmusi...	1	1	EC2	--

Choose the link to Amazon ECS running on your ECS cluster

5. Select the **Target Group Name** link.



Service : a2c-iis-mvcmusicstore-30f1a47ead48-ECS-LBWebAppStack-OSCZHZ6827U0-Service-rMgQExiYp8SS

Cluster	a2c-iis-mvcmusicstore-30f1a47ead48-ECS-Cluster	Desired count	1
Status	ACTIVE	Pending count	0
Task definition	a2c-iis-mvcmusicstore-30f1a47ead48-ECS-iis-mvcmusicstore-30f1a47ead48:1	Running count	1
Service type	REPLICA		
Launch type	EC2		
Service role	AWSServiceRoleForECS		
Created By	arn:aws:iam::[redacted]:user/app2container		

Load Balancing

Target Group Name	Container Name	Container Port
a2c-i-Targe-Q43WN9ELI88W	iis-mvcmusicstore-30f1a47ead48	8081

Network Access

Health check grace period 60

Service discovery

Service discovery endpoint iis-mvcmusicstore-30f1a47ead48.a2c-iis-mvcmusicstore-30f1a47ead48-ECS.local

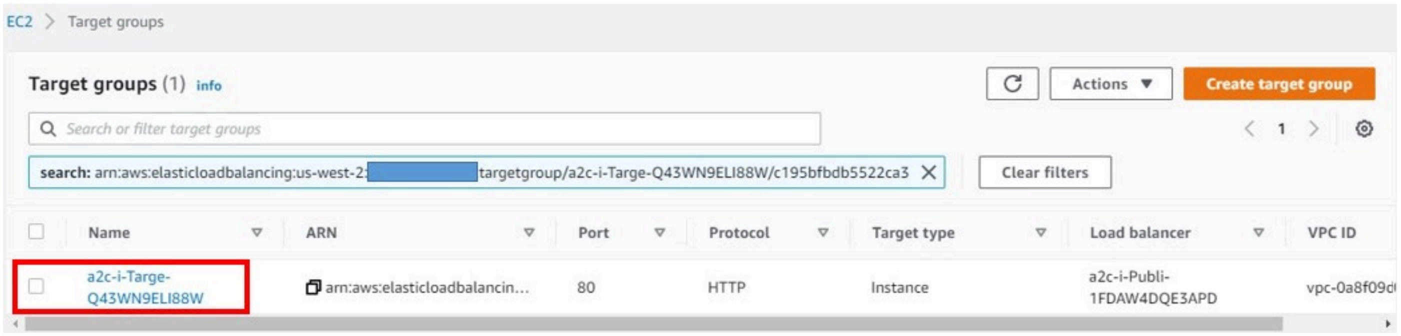
Service discovery name iis-mvcmusicstore-30f1a47ead48

DNS record type ...	Container...	TTL
SRV	iis-mvcmu...	60

Namespace a2c-iis-mvcmusicstore-30f1a47ead48-ECS.local (PRIVATE)

*Select the **Target Group Name** link*

6. Select the target group.

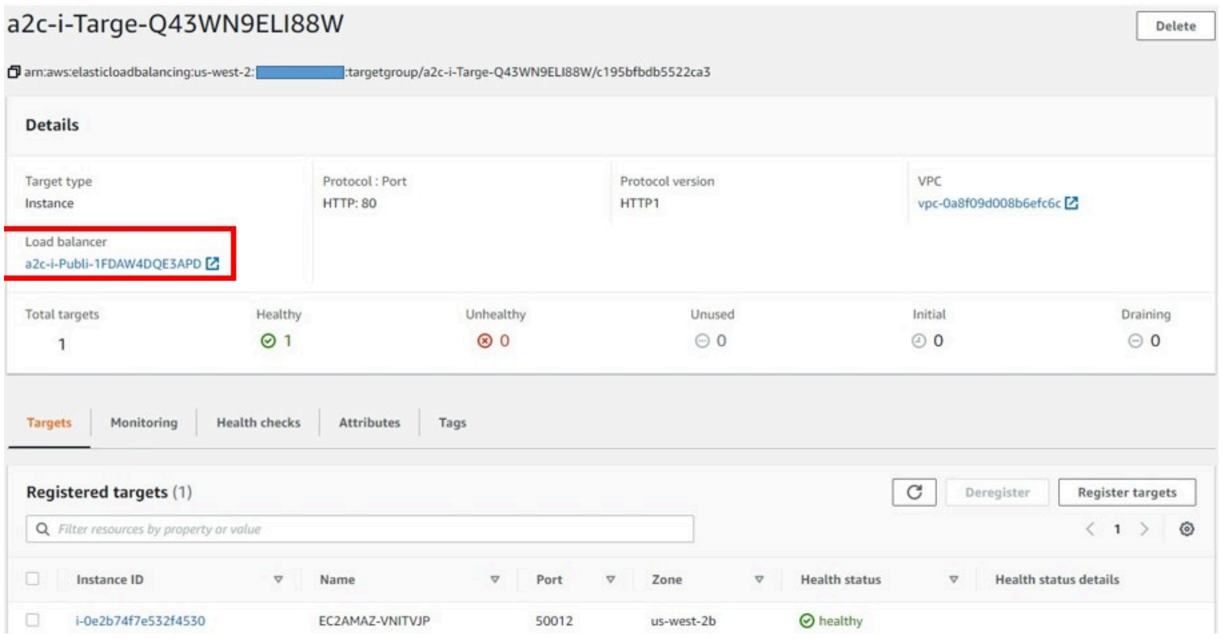


search: arn:aws:elasticloadbalancing:us-west-2:[redacted]targetgroup/a2c-i-Targe-Q43WN9ELI88W/c195bfbdb5522ca3

Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
a2c-i-Targe-Q43WN9ELI88W	arn:aws:elasticloadbalancin...	80	HTTP	Instance	a2c-i-Publi-1FDAW4DQE3APD	vpc-0a8f09d

Select the target group

7. Select the **Load Balancers details** link.



a2c-i-Targe-Q43WN9ELI88W

Delete

arn:aws:elasticloadbalancing:us-west-2:[redacted]:targetgroup/a2c-i-Targe-Q43WN9ELI88W/c195bfbdb5522ca3

Details

Target type	Protocol : Port	Protocol version	VPC
Instance	HTTP: 80	HTTP1	vpc-0a8f09d008b6efc6c

Load balancer
a2c-i-Publi-1FDAW4DQE3APD

Total targets	Healthy	Unhealthy	Unused	Initial	Draining
1	1	0	0	0	0

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (1)

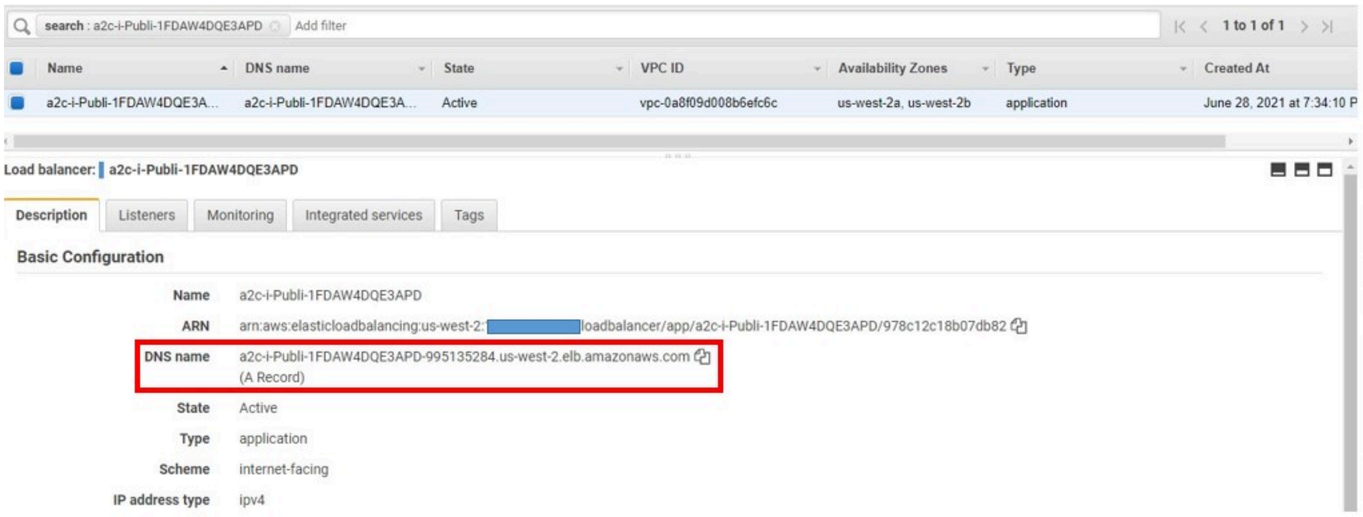
Deregister | Register targets

Filter resources by property or value

Instance ID	Name	Port	Zone	Health status	Health status details
i-0e2b74f7e532f4530	EC2AMAZ-VNITVJP	50012	us-west-2b	healthy	

*Select the **Load Balancers details** link*

8. On the **Load Balancers** page, note the DNS name in the **Basic Configuration** details. Copy and paste the DNS link into your browser to view your containerized application running in Amazon ECS.



search : a2c-i-Publi-1FDAW4DQE3APD   Add filter   1 to 1 of 1

Name	DNS name	State	VPC ID	Availability Zones	Type	Created At
a2c-i-Publi-1FDAW4DQE3A...	a2c-i-Publi-1FDAW4DQE3A...	Active	vpc-0a8f09d008b6efc6c	us-west-2a, us-west-2b	application	June 28, 2021 at 7:34:10 P

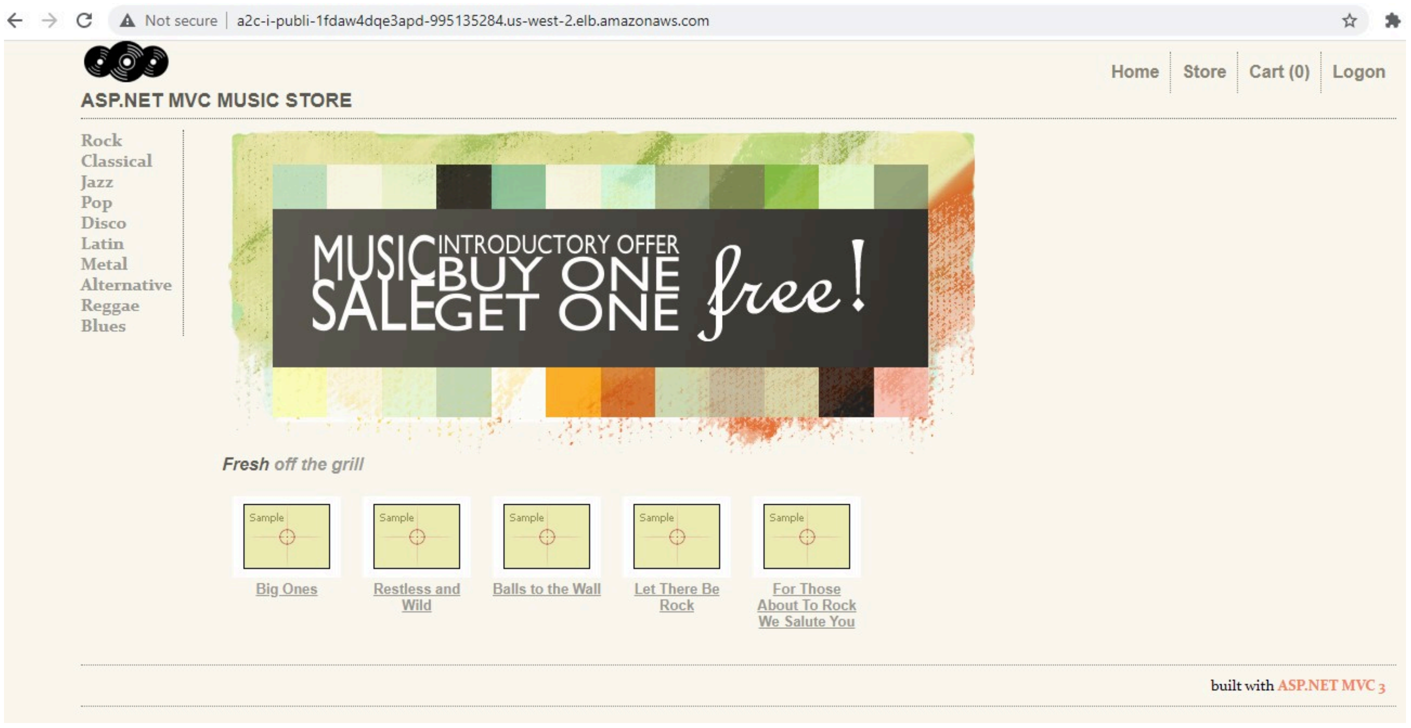
Load balancer: a2c-i-Publi-1FDAW4DQE3APD

Description | Listeners | Monitoring | Integrated services | Tags

Basic Configuration

Name	a2c-i-Publi-1FDAW4DQE3APD
ARN	arn:aws:elasticloadbalancing:us-west-2:[redacted]loadbalancer/app/a2c-i-Publi-1FDAW4DQE3APD/978c12c18b07db82
DNS name	a2c-i-Publi-1FDAW4DQE3APD-995135284.us-west-2.elb.amazonaws.com (A Record)
State	Active
Type	application
Scheme	internet-facing
IP address type	ipv4

Copy and paste the DNS link into your browser to view your containerized application running in Amazon ECS

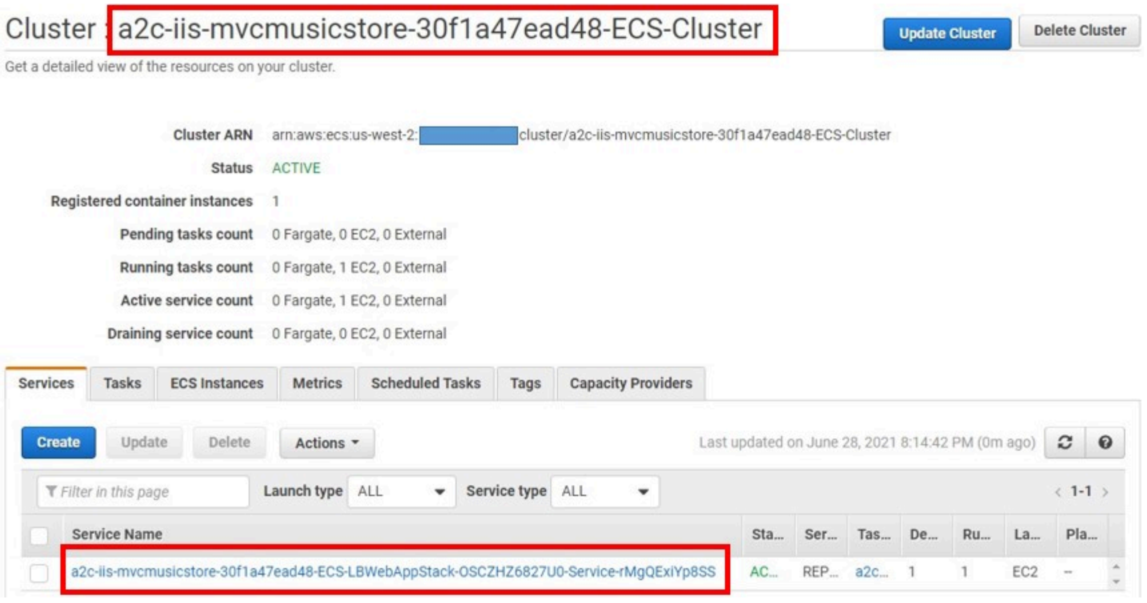


Containerized application

Now that you have your application up and running, you will create a CI/CD pipeline for it using App2Container.

To create a CI/CD pipeline:

1. Navigate to Amazon ECS in the AWS Management Console and copy your **Cluster** name and **Service** name to your local machine.



Cluster : a2c-iis-mvcmusicstore-30f1a47ead48-ECS-Cluster

Get a detailed view of the resources on your cluster.

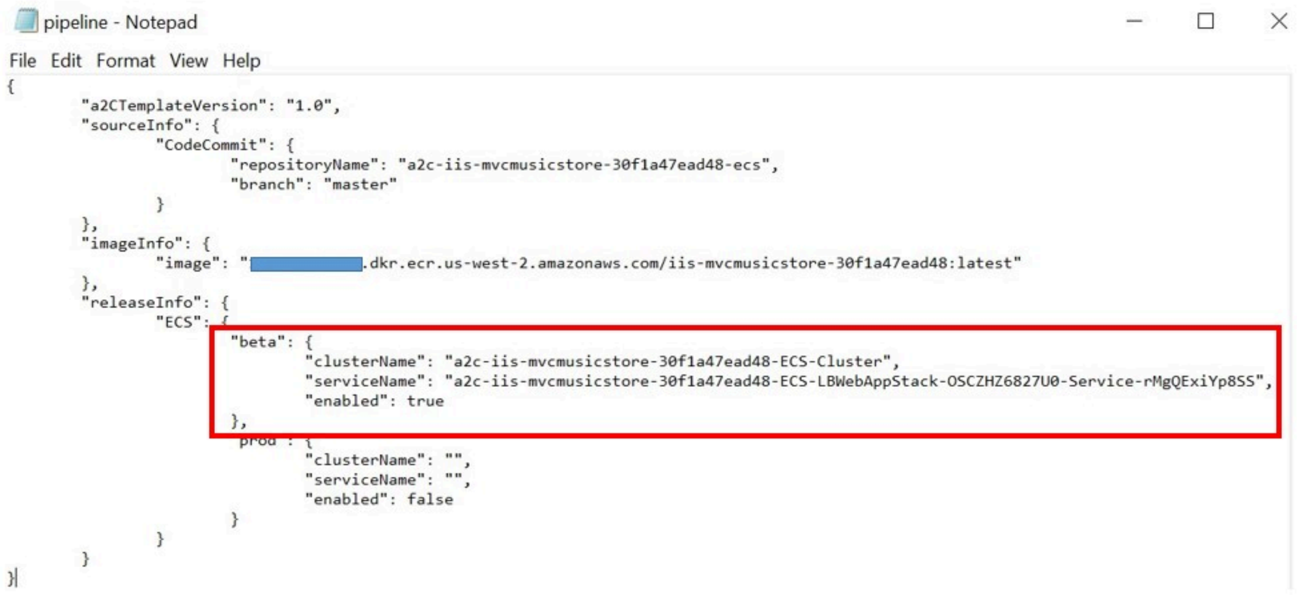
Cluster ARN	arn:aws:ecs:us-west-2:[redacted]cluster/a2c-iis-mvcmusicstore-30f1a47ead48-ECS-Cluster
Status	ACTIVE
Registered container instances	1
Pending tasks count	0 Fargate, 0 EC2, 0 External
Running tasks count	0 Fargate, 1 EC2, 0 External
Active service count	0 Fargate, 1 EC2, 0 External
Draining service count	0 Fargate, 0 EC2, 0 External

Last updated on June 28, 2021 8:14:42 PM (0m ago)

Service Name	Sta...	Ser...	Tas...	De...	Ru...	La...	Pla...
a2c-iis-mvcmusicstore-30f1a47ead48-ECS-LBWebAppStack-OSCZHZ6827U0-Service-rMgQExiYp8SS	AC...	REP...	a2c...	1	1	EC2	--

*Copy your **Cluster** name and **Service** name to your local machine*

2. On your worker machine, open the `pipeline.json` file at `C:\Users\Administrator\AppData\Local\app2container\<net-app-id>\` and fill in the values for the `beta.clusterName` and `beta.serviceName` based on your deployment. Additionally, change `beta.enabled` to **`true`**, and save the file.



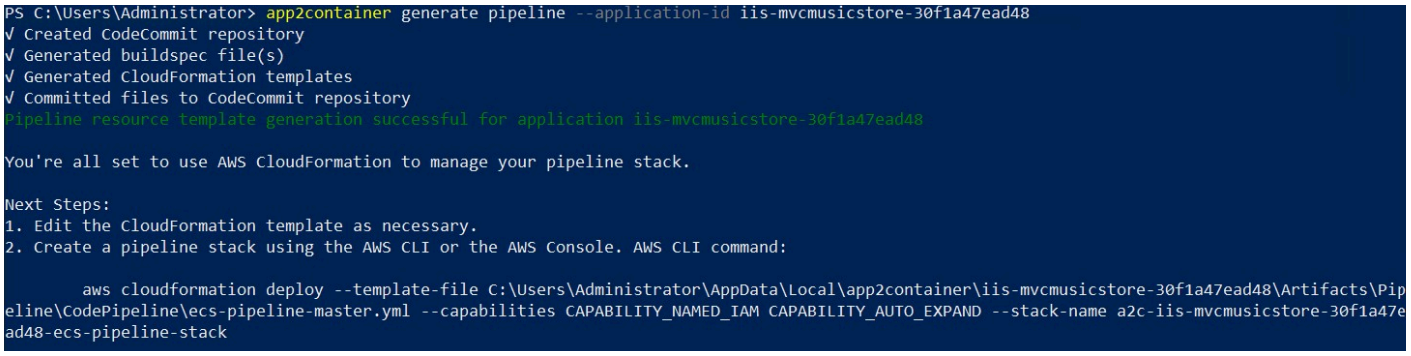
```
{
        "a2CTemplateVersion": "1.0",
        "sourceInfo": {
                "CodeCommit": {
                        "repositoryName": "a2c-iis-mvcmusicstore-30f1a47ead48-ecs",
                        "branch": "master"
                }
        },
        "imageInfo": {
                "image": "[redacted].dkr.ecr.us-west-2.amazonaws.com/iis-mvcmusicstore-30f1a47ead48:latest"
        },
        "releaseInfo": {
                "ECS": {
                        "beta": {
                                "clusterName": "a2c-iis-mvcmusicstore-30f1a47ead48-ECS-Cluster",
                                "serviceName": "a2c-iis-mvcmusicstore-30f1a47ead48-ECS-LBWebAppStack-OSCZHZ6827U0-Service-rMgQExiYp8SS",
                                "enabled": true
                        },
                        "prod": {
                                "clusterName": "",
                                "serviceName": "",
                                "enabled": false
                        }
                }
        }
}
```

Open the `pipeline.json` file

3. Open PowerShell and run the `app2container generate pipeline` command as follows:

```
app2container generate pipeline --application-id <net-app-id>
```

4. You should see the following output:



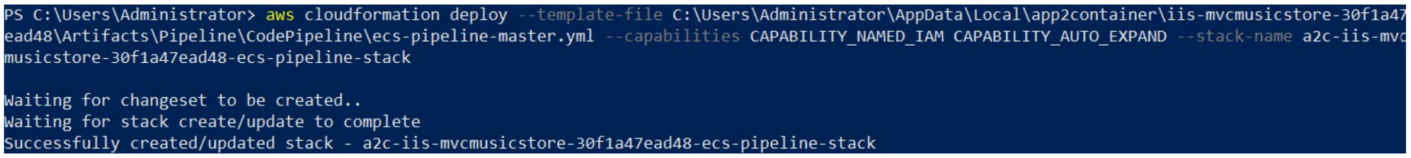
```
PS C:\Users\Administrator> app2container generate pipeline --application-id iis-mvcmusicstore-30f1a47ead48
√ Created CodeCommit repository
√ Generated buildspec file(s)
√ Generated CloudFormation templates
√ Committed files to CodeCommit repository
Pipeline resource template generation successful for application iis-mvcmusicstore-30f1a47ead48

You're all set to use AWS CloudFormation to manage your pipeline stack.

Next Steps:
1. Edit the CloudFormation template as necessary.
2. Create a pipeline stack using the AWS CLI or the AWS Console. AWS CLI command:

        aws cloudformation deploy --template-file C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47ead48\Artifacts\Pip
eline\CodePipeline\ecs-pipeline-master.yml --capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND --stack-name a2c-iis-mvcmusicstore-30f1a47e
ad48-ecs-pipeline-stack
```

PowerShell output

5. Run the suggested `aws cloudformation deploy` command to deploy your pipeline to AWS.

```
aws cloudformation deploy --template-file
C:\Users\Administrator\AppData\Local\app2container\<net-app-id>\
Artifacts\Pipeline\CodePipeline\ecs-pipeline-master.yml
--capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND
--stack-name a2c-<net-app-id>-ecs-pipeline-stack
```

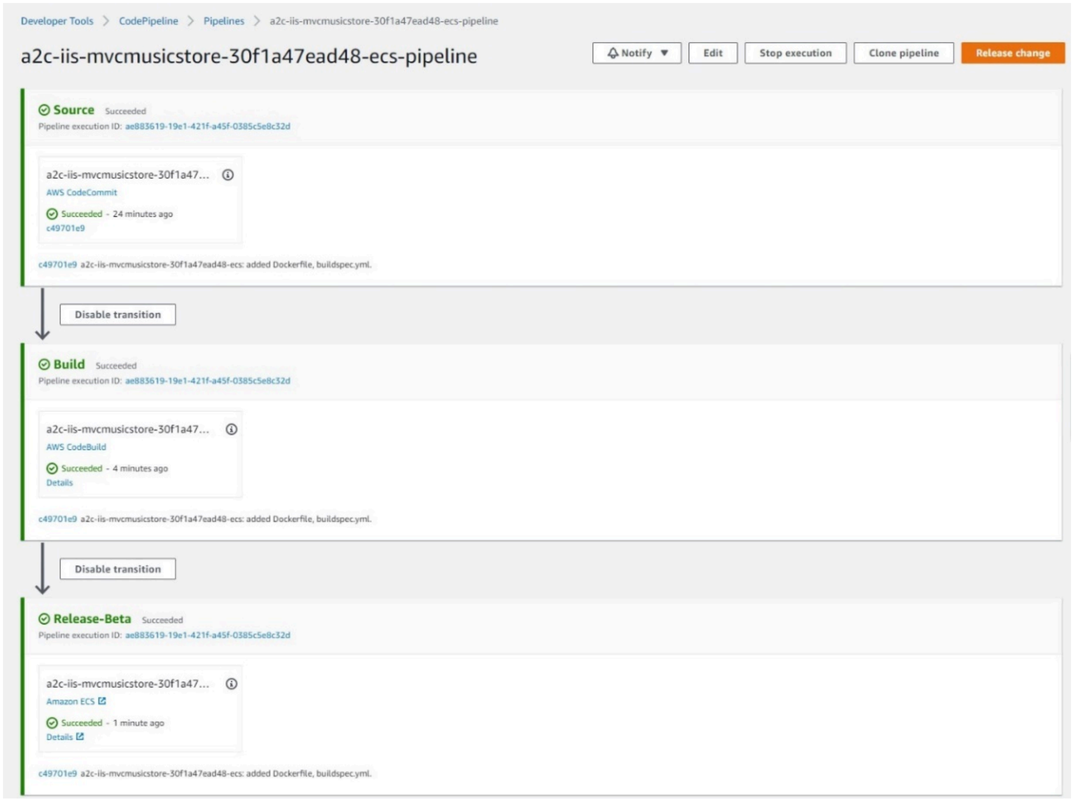
This will take several minutes, because the command will not complete until the pipeline phases complete. Part of this is a build phase, which requires a build of the application (and Windows Server base image). You can monitor the progress in the CloudFormation service in the AWS Management Console. You will see the following output upon completion:



```
PS C:\Users\Administrator> aws cloudformation deploy --template-file C:\Users\Administrator\AppData\Local\app2container\iis-mvcmusicstore-30f1a47
ead48\Artifacts\Pipeline\CodePipeline\ecs-pipeline-master.yml --capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND --stack-name a2c-iis-mvc
musicstore-30f1a47ead48-ecs-pipeline-stack

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - a2c-iis-mvcmusicstore-30f1a47ead48-ecs-pipeline-stack
```

Successful creation of the pipeline using CloudFormation

To confirm that the pipeline was successfully created, go to the CodePipeline service in the AWS Management Console and choose your newly created pipeline.


Developer Tools > CodePipeline > Pipelines > a2c-iis-mvcmusicstore-30f1a47ead48-ecs-pipeline

a2c-iis-mvcmusicstore-30f1a47ead48-ecs-pipeline

Source Succeeded
Pipeline execution ID: ae883619-19e1-421f-a45f-0385c5e8c32d

a2c-iis-mvcmusicstore-30f1a47...
AWS CodeCommit
Succeeded - 24 minutes ago
c49701e9

c49701e9 a2c-iis-mvcmusicstore-30f1a47ead48-ecs: added Dockerfile, buildspec.yml.

Build Succeeded
Pipeline execution ID: ae883619-19e1-421f-a45f-0385c5e8c32d

a2c-iis-mvcmusicstore-30f1a47...
AWS CodeBuild
Succeeded - 4 minutes ago
Details

c49701e9 a2c-iis-mvcmusicstore-30f1a47ead48-ecs: added Dockerfile, buildspec.yml.

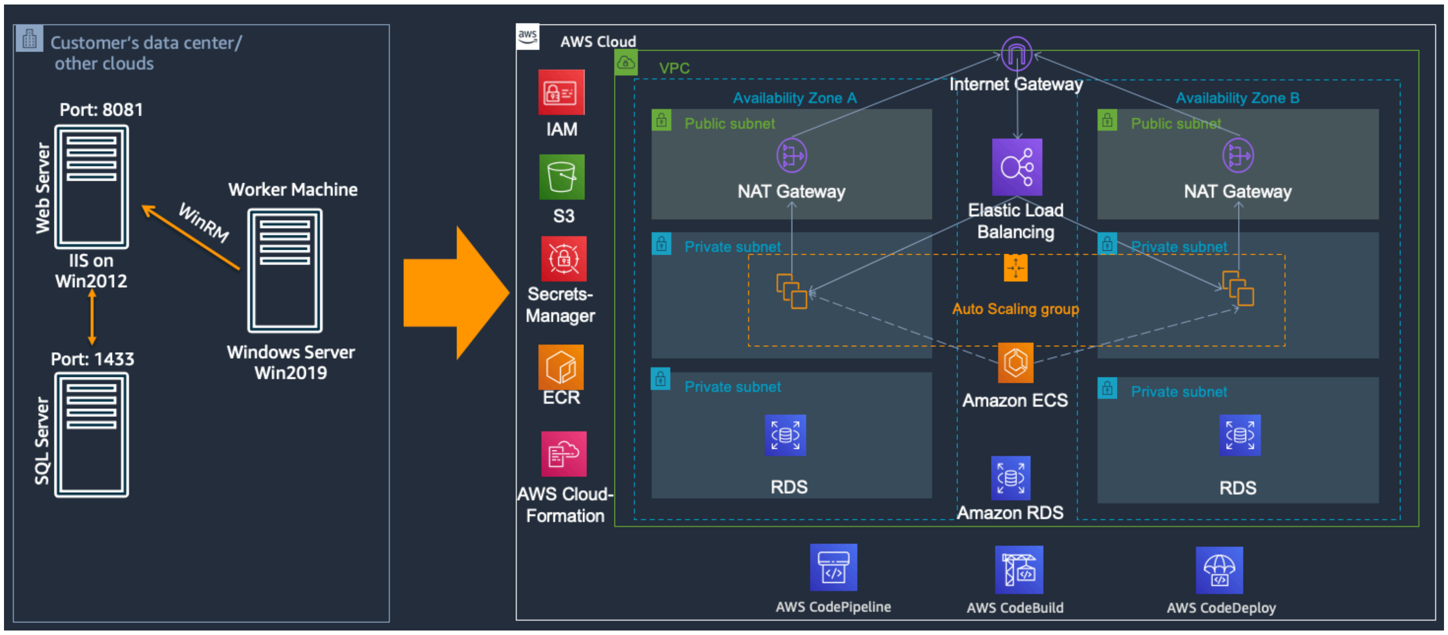
Release-Beta Succeeded
Pipeline execution ID: ae883619-19e1-421f-a45f-0385c5e8c32d

a2c-iis-mvcmusicstore-30f1a47...
Amazon ECS
Succeeded - 1 minute ago
Details

c49701e9 a2c-iis-mvcmusicstore-30f1a47ead48-ecs: added Dockerfile, buildspec.yml.

Confirm successful completion of newly created pipeline

You have successfully containerized a Windows VM-based application and deployed it on Amazon ECS with a fully automated CI/CD pipeline. The following figure shows the architecture for this setup. If you are interested in doing a migration of the SQL Server instance running on Amazon EC2 to Amazon Relational Database Service, refer to the Amazon RDS for SQL Server Workshop.



Architecture for containerizing a Windows VM-based application and deploying it on Amazon ECS with a fully automated CI/CD pipeline

Logging and monitoring

Monitoring is an important part of maintaining the reliability, availability, and performance of your applications running on Amazon ECS. Collect monitoring data from all of the parts of your AWS stack so you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon ECS resources and responding to potential incidents. Refer to the following table for details on each service.

Table 7 — Monitoring services

Service	Description
Amazon CloudWatch Alarms	For clusters with tasks or services using the EC2 launch type, you can use CloudWatch Alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation.
Amazon CloudWatch Logs	Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the `awslogs` log driver in your task definitions. This is the only supported method for accessing logs for tasks using the Fargate launch type, but also works with tasks using the EC2 launch type.
Amazon CloudWatch Events	Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action.
AWS CloudTrail	CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon ECS. Using the information collected by CloudTrail, you can determine the request that was made to Amazon ECS, the IP address from which the request was made, who made the

Service	Description
	request, when it was made, and additional details.
AWS Trusted Advisor	Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps.
Amazon ECS events and Amazon EventBridge	Amazon ECS events for EventBridge receives near real-time notifications regarding the current state of your Amazon ECS clusters. If your tasks are using the Fargate launch type, you can see the state of your tasks. If your tasks are using the EC2 launch type, you can see the state of both the container instances and the current state of all tasks running on those container instances. For services, you can see events related to the health of your service.

Service	Description
AWS X-Ray	AWS X-Ray helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture. With X-Ray, you can understan d how your application and its underlyin g services are performing to identify and troubleshoot the root cause of performance issues and errors. You can use the X-Ray SDK and AWS service integration to instrument requests to your applications that are running locally or on AWS compute services such as Amazon EC2, AWS Elastic Beanstalk, Amazon ECS, and AWS Lambda.

Security

When you migrate your .NET applications to containers, many of the same security concerns as non-containerized environments need to be addressed. This section walks through the primary security components of running containerized applications in AWS. It covers how to use Windows authentication for user and application authentication, and how to control access of the containerized applications to other AWS services using IAM.

Topics

Windows authentication

Many .NET applications use Windows (or integrated) authentication to enable users to sign in using their Active Directory domain credentials. Applications can also utilize Active Directory service accounts to connect to network resources such as SQL Server databases. If an application running on an EC2 instance (application server) needs access to a SQL Server database running on another EC2 instance (database server), both EC2 instances need to either join the same domain or join different domains with a trust relationship between them. Windows containers cannot be joined by domain, but they can be configured to use Active Directory identities.

To enable an application running inside a container to authenticate against a domain:

1. Join the ECS instance (host) to the domain.
2. Configure a group Managed Service Account (gMSA) in the domain.
3. Change your Dockerfile to support windows authentication.

4. Create the `CredentialSpec` file.
5. Configure the ECS Task Definition with `CredentialSpec`.

Join the ECS instance (host) to the domain

There are many ways to join the ECS instance to an Active Directory domain. It can be done manually (by connecting to the instance through RDP) or automatically. AWS enables you to save costs by automatically reducing compute capacity in times of low demand, and provision more capacity when demand increases. To take advantage of this elasticity, a best practice is to use Auto Scaling groups for provisioning ECS instances.

The User Data section in the launch template/configuration that is used with the Auto Scaling group can include domain join commands. If your Active Directory domain is based on AWS Directory Service or you use AD Connector to connect to an on-premises Active Directory domain, you can use AWS Systems Manager Run Command and run the AWS-JoinDirectoryServiceDomain document. There are two prerequisites to using this approach that are described as follows.

1. If the ECS instances and the Active Directory domain are provisioned in different VPCs, make sure they that the VPCs can communicate through VPC peering or transit gateway.
2. The ECS instances need permissions (through IAM policies) to communicate to the Systems Manager and Directory Service APIs. AWS recommends creating custom policies that take into account your system needs and security requirements. However, as a starting point, you can use the following policies:
   - `AmazonSSMManagedInstanceCore` — This AWS managed policy enables an instance to use Systems Manager service core functionality.
   - `AmazonSSMDirectoryServiceAccess` — This AWS managed policy allows AWS Systems Manager Agent (SSM Agent) to access AWS Directory Service on your behalf for requests to join the Active Directory domain by the managed instance.

Configure a group Managed Service Account in the Active Directory domain

A group Managed Service Account (gMSA) is a type of service account available in Windows Server 2012 and later. When a container is configured to use a gMSA, it does not know the password for the account. The gMSA password is configured on the Active Directory domain controller. When a container using gMSA runs on a domain-joined ECS instance, the ECS instance retrieves the password for the gMSA from the Active Directory domain controller and passes it to the container. It is recommended to create a security group for each gMSA account and add ECS instances (that will use the gMSA account) to the security group. This limits access to the gMSA to only the ECS instances that need it.

The following PowerShell snippet demonstrates how to create a security group, create a gMSA, and add the ECS Container Instance to the security group. To run the following commands, you will need to use an account that belongs to the Domain Admins security group or has been delegated the Create the `msDS-GroupManagedServiceAccount` objects permission. On the machine that you are running these commands on, you will need to install Remote Server Administration Tools.

```
# Create the AD group
New-ADGroup -Name "Amazon ECS Authorized Container instances"
-SamAccountName "ECSContainerInstances" -GroupScope DomainLocal

# Create the gMSA
New-ADServiceAccount -Name "gmsaecs" -DnsHostName "gmsaecs.YOURDOMAIN_FQDN"
-ServicePrincipalNames "host/gmsaecs", "host/gmsaecs.YOURDOMAIN_FQDN"
-PrincipalsAllowedToRetrieveManagedPassword "ECSContainerInstances"

# Add your ECS Container Instance to the AD group
Add-ADGroupMember -Identity "ECSContainerInstances"
-Members "ECSContainerInstances01$", "ECSContainerInstances02$",
"ECSContainerInstances03$"
```

Change your Dockerfile to support Windows authentication

The following snippet demonstrates how to configure your IIS application running inside a container to use a gMSA. The following Dockerfile instructions install and configure Windows authentication inside the container, and on IIS.

```
# Install Windows Auth in IIS Feature
RUN Install-WindowsFeature -Name Web-Windows-Auth –IncludeAllSubFeature

# Configure the IIS Application Pool account to use Network Service account.
That enables it to leverage gMSA.
RUN Import-Module WebAdministration; Set-ItemProperty IIS:\AppPools\SiteName
-name processModel.identityType -value 2
```

```
# Disable Anonymous authentication on IIS
RUN Import-Module WebAdministration; Set-WebConfigurationProperty
-Filter '/system.webServer/security/authentication/anonymousAuthentication'
-Name Enabled -Value False -PSPath 'IIS:\' -Location 'SiteName'

# Enable Windows Authentication
RUN Import-Module WebAdministration; Set-WebConfigurationProperty
-Filter '/system.webServer/security/authentication/windowsAuthentication'
-Name Enabled -Value True -PSPath 'IIS:\' -Location 'SiteName'
```

Create the CredentialSpec file

In the previous sections, you joined your ECS instance to an Active Directory domain, created a gMSA, and configured your application to use the gMSA. In this section, you will configure your ECS Task Definition to use the gMSA using a credential spec file. A credential spec file is a JSON document that contains metadata about your gMSA account.

To create the credential spec file, you can run the following PowerShell cmdlets. You will need to run these cmdlets on either the ECS instance, or a domain joined EC2 instance that has the RSAT AS PowerShell tools installed. For more information on how to create a credential spec file, refer to the Create gMSAs for Windows containers documentation.

```
#Install the CredentialSpec module
Install-Module CredentialSpec

#Create a credential spec using the gMSA name at the provided path
New-CredentialSpec -AccountName gmsaecs -Path "C:\gmsa\gmsaecs_credspec.json"
```

The following snippet shows an example credential spec file.

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "gmsaecs",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "gmsaecs",
                "Scope": "example.com"
            }
        ]
    }
}
```

Configure ECS Task Definition with CredentialSpec

After you have created the credential spec file, you will reference it in your ECS task definition. Keeping the gMSA metadata in a credential spec file (separate from the Dockerfile) gives you the ability to change the gMSA used by your application in the future without the need for code changes in the application.

Amazon ECS supports the following ways to reference the credential spec file in the **dockerSecurityOptions** field of a task definition. For more information and example task definitions, refer to the Amazon ECS documentation.

Add the credential spec to an Amazon S3 bucket, then reference the ARN of the Amazon S3 bucket in the **dockerSecurityOptions** field of the task definition. Your Amazon ECS task execution role must have the `s3:Get*` and `s3:List*` permissions on your S3 bucket and the credential spec object.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "dockerSecurityOptions": [
                "credentialspec:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
            }
        ],
        ...
}
```

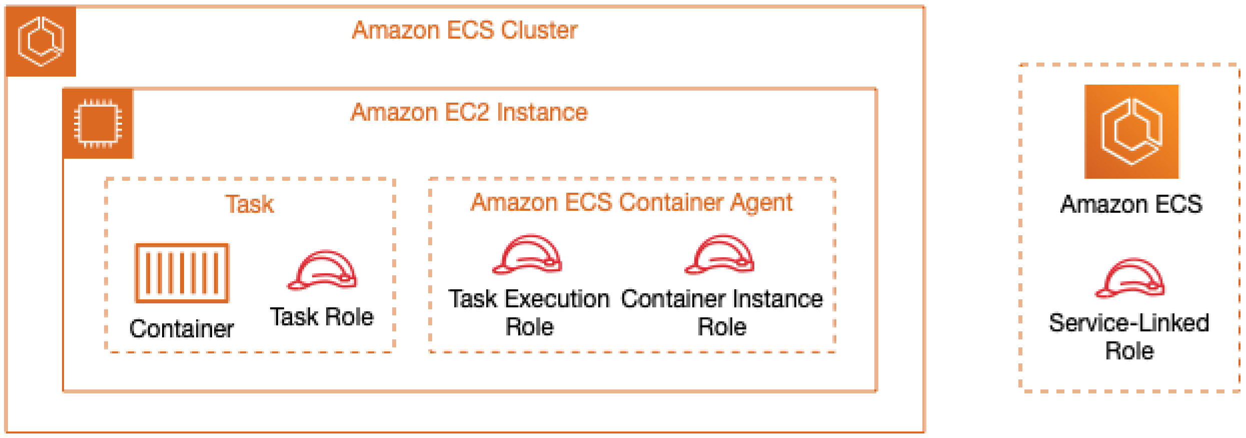
Using a load balancer with Windows Authentication

A typical architecture for a containerized ASP.NET application would involve ELB. A load balancer automatically distributes incoming traffic across multiple targets such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. Windows Authentication requires that the source port be preserved in the connection from the client to the server. A Network Load Balancer with a TCP listener will preserve the source port for a load balanced connection. For that reason, use a Network Load Balancer when using Windows Authentication.

This section covered how to enable users to sign in using their Active Directory domain credentials, and how your applications can also utilize Active Directory service accounts to connect to network resources, such as SQL Server databases. When running containers, you also must consider access control to AWS resources that may occur during container related lifecycle activities. IAM helps you securely control access granted to Amazon ECS, Amazon ECS container agent, and your application during these events.

Authenticating with AWS services

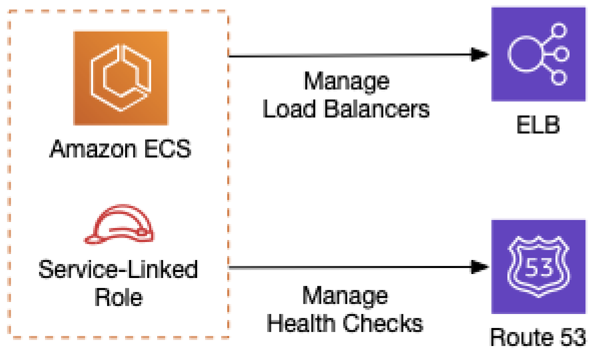
Amazon ECS provides multiple levels to secure your applications and clusters using IAM while containerized applications interact with other AWS services. The following diagram depicts the types of roles Amazon ECS supports when using the EC2 launch type.



Types of roles Amazon ECS supports when using the EC2 launch type

Service-linked role

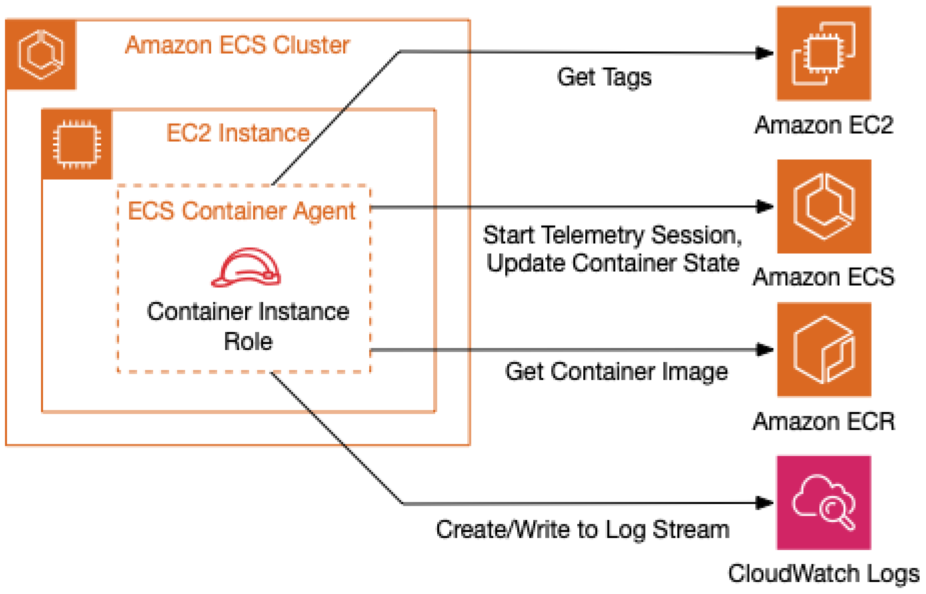
There are multiple activities that the Amazon ECS runs while it orchestrates your container workloads. Amazon ECS uses a service-linked role for the permissions it requires to call other AWS services on your behalf. These include services such as Amazon EC2 to manage elastic network interfaces, ELB to manage targets, and Amazon Route 53 for creating health checks. A more detailed list can be found on the Service-linked role for Amazon ECS page.



Amazon ECS service-linked role architecture

Container instance role

The container instance role is the IAM role used as the Instance role by EC2 instances running your containers. This role is also used by the Amazon ECS container agent to make calls to AWS services and connect with the Amazon ECS to register container instances, report status, and get commands. Other examples include the agent starting a telemetry session, or creating the Amazon ECS cluster if one does not already exist. A more detailed list can be found on the Amazon ECS container instance IAM role page.



Container instance role architecture

Task execution role

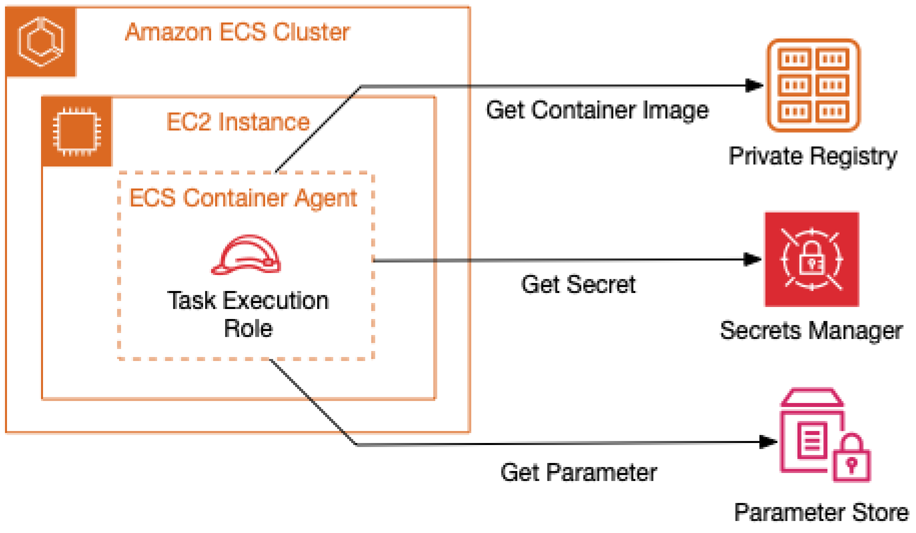
The task execution role grants the Amazon ECS container agent permission to make AWS API calls on your behalf when an Amazon ECS task is started. An example of an activity that the Amazon ECS Agent runs during this time is pulling container images from a private repository, and private registry authentication needs to be configured.

Another use case where this role is required is injecting sensitive data into your containers. You might choose to store sensitive data (such as database connection strings) in either AWS Secrets Manager or AWS Systems Manager Parameter Store, and reference them in your container definition. Sensitive data is injected into your container as environment variables when the container is initially started without having to write code to retrieve the values.

If the secret or Parameter Store parameter is subsequently updated or rotated, the container will not receive the updated value automatically. Either launch a new task, or if your task is part of a service, you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.

For Windows tasks that are configured to use the `awslogs` logging driver, set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. This can be done with user data using the following syntax:

```
<PowerShell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster '<cluster name>' -LoggingDrivers '["json-file","awslogs"]'
 -EnableTaskIAMRole
</PowerShell>
```



Task execution role architecture

Task role

The task role is the IAM role assigned to the Containers instances created as part of the Amazon ECS task. This role provides applications using the AWS SDK or CLI the AWS credentials used to make API requests to authorized AWS services. For example, you can set the policy associated with the task role to allow your application to read/write items from or to DynamoDB, publish an event to an EventBridge bus, or start an AWS Step Functions workflow.

One of the requirements that must be met to enable IAM roles for tasks on Windows is that the `EnableTaskIAMRole` option be set when you launch an EC2 instance. This can be done by using a user data script. For example:

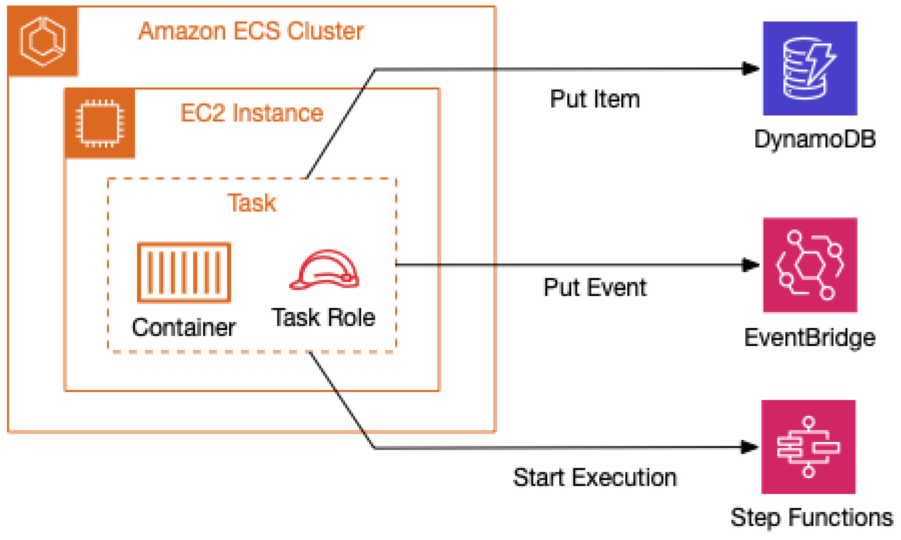
```
<PowerShell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster '<cluster name>' -EnableTaskIAMRole
</PowerShell>
```

Additionally, before containers can access the credential proxy on the container instance to get credentials, the container instance must be bootstrapped with the required networking commands. The following code example script should be run on your containers when they start.

```
$gateway = (Get-NetRoute | Where { $_.DestinationPrefix -eq '0.0.0.0/0' } | Sort-Object
 RouteMetric | Select NextHop).NextHop
$ifIndex = (Get-NetAdapter -InterfaceDescription "Hyper-V Virtual Ethernet*" | Sort-
Object | Select ifIndex).ifIndex
New-NetRoute -DestinationPrefix 169.254.170.2/32 -InterfaceIndex $ifIndex -NextHop
 $gateway # credentials API
New-NetRoute -DestinationPrefix 169.254.169.254/32 -InterfaceIndex $ifIndex -NextHop
 $gateway # metadata API
```

For a complete list of requirements, refer to the Windows IAM roles for tasks page on the Amazon ECS Developer Guide.

Now that you know how to enable IAM roles for Amazon ECS Tasks on Windows, learn how your application code can assume the role.



Task role architecture

AWS SDK for .NET credential loading

The AWS SDKs take the complexity out of coding by providing language-specific APIs for AWS services. One of the ways the AWS SDK for .NET helps is by seamlessly loading the IAM credentials at runtime. The AWS SDK for .NET searches for credentials in a certain order and uses the first available set for the current application. In its simplest form, credential loading is transparent, as can be seen by the following code example that lists the objects in an S3 bucket.

```
var s3Client = new AmazonS3Client();
var listRequest = new ListObjectsRequest
{
    BucketName = "SampleBucket",
};

ListObjectsResponse response = await s3Client.ListObjectsAsync(listRequest);

foreach (var item in response.S3Objects)
{
    Console.WriteLine("key = {0} size = {1}", item.Key, item.Size);
}
```

The parameterless constructors for AWS service clients (AmazonS3Client in this example) rely on the FallbackCredentialsFactory to load the credentials using a well-known credential and profile resolution order and includes support for Amazon ECS. This provides a frictionless developer experience while still ensuring credentials remain secure.

This section covered the various IAM roles involved in a container's lifecycle, including how to securely retrieve temporary credentials using the AWS SDK for .NET. Your security journey does not end here. Consider how to keep data secure both in transit and at rest.

In-flight data protection using encryption

By default, API calls to Amazon ECS travel through the public internet. To keep that traffic within the AWS global network, you can configure Amazon ECS to use an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Amazon ECS APIs by using private IP addresses. PrivateLink restricts all network traffic between your VPC and Amazon ECS to the Amazon network. You don't need an internet gateway, NAT device, or virtual private gateway.

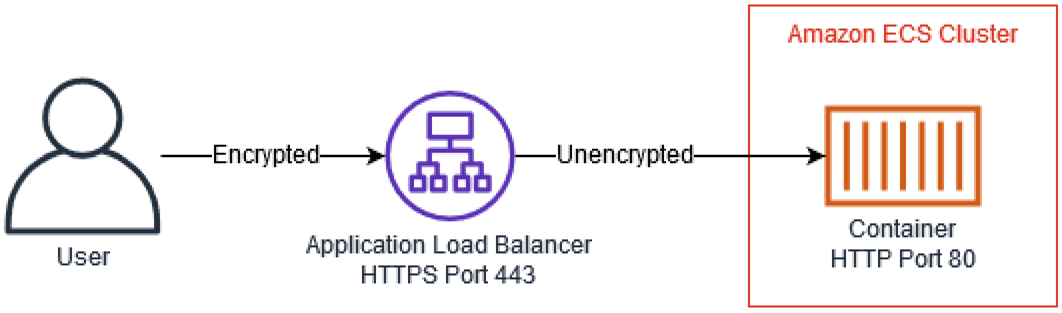
You can create VPC endpoints for Amazon ECS, Amazon ECS container agent, and Amazon ECS telemetry in the Region where your containers are deployed. VPC endpoints currently do not support cross-Region requests, so if you have a multi-Region deployment, consider following the recommendations in the Integrating cross VPC ECS cluster for enhanced security with AWS App Mesh blog post. For more information on building a scalable multi-Region architecture in AWS, refer to the Building a Scalable and Secure Multi-VPC AWS Network Infrastructure whitepaper. If you are using Amazon ECS' integration with Secrets Manager or Systems Manager Parameter Store for sensitive data, you will also need to configure VPC endpoints for each of these services.

In addition to securing network traffic by restricting it to the AWS network, you can encrypt data in transit between your application and AWS services by enforcing the use of TLS 1.2 when using the AWS SDK for .NET. The following sections review the various approaches to using encryption in transit for applications running on Amazon ECS.

TLS termination at the load balancer

It's a best practice to enforce TLS termination at the load balancer and both Application Load Balancers and Network Load Balancers support TLS termination. Terminating TLS connections at the load balancer frees up your backend containers from the work of encrypting and decrypting your traffic. Your containers handle plain HTTP requests while offloading the complexity of managing HTTPS connections to the Load Balancers.

This approach also simplifies certificate management since the certificates are now deployed to the load balancers instead of backend containers. Additionally, you can use AWS Certificate Manager at no charge to securely store, expire, rotate, and update your certificates. This process involves adding a TLS listener to your load balancer, configuring the backend container to listen on an unencrypted port, such as part 80 (HTTP), and configuring the listener on the load balancer to forward traffic to the unencrypted port used by your container. Refer to the TLS Termination for Network Load Balancers blog post for more information.



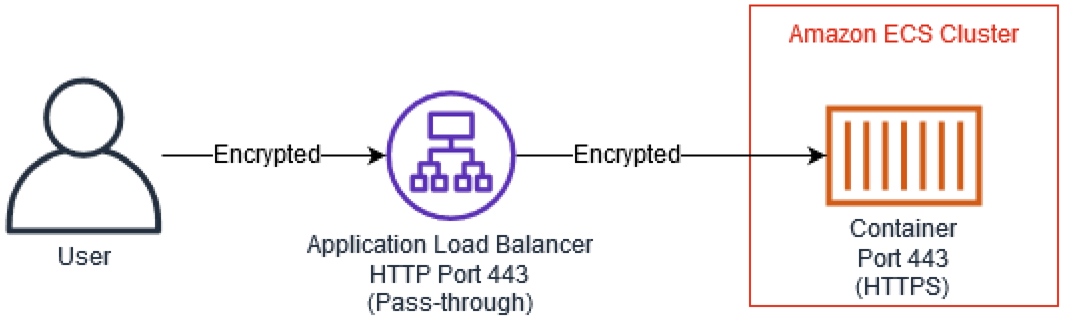
TLS termination at Application Load Balancer

End-to-end encryption

Terminating TLS connections at the load balancer and using HTTP on the backend may be sufficient for your application. However, if you are developing an application that needs to comply with strict external regulations, you may be required to secure all network connections. You can configure the load balancer to either pass TLS traffic through untouched (terminate TLS at container), or decrypt and re-encrypt for end-to-end encryption.

Terminate TLS at the container level

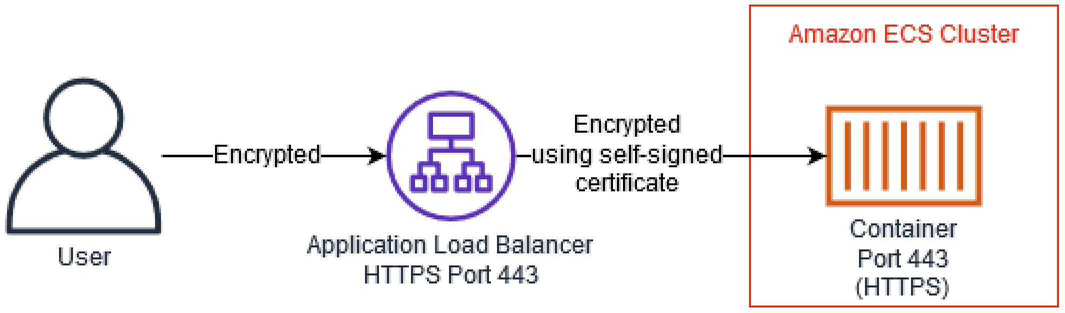
This process involves adding an unencrypted listener to your load balancer, configuring backend containers to listen on the secure port and terminate HTTPS connections, and configuring the listener on the load balancer to forward traffic to the secure port used by the backend containers.



TLS termination at the container level

Decrypt and re-encrypt

This process involves adding a TLS listener to your load balancer, configuring backend containers to listen on the secure port, terminate HTTPS connections, using a self-signed certificate, and configuring the listener on the load balancer to forward traffic to the secure port used by the backend containers.



Re-encrypt traffic using self-signed certificate

Source code

The source code used in this guide is hosted on GitHub at aws-samples/dotnet-modernization-music-store. The starting point for the replatforming walkthrough is the main branch that we containerize and deploy to Amazon ECS using AWS App2Container.

Conclusion

This guide describes the business and technical aspects of replatforming an existing .NET Framework application to Windows containers. Anyone tasked with evaluating modernization of Windows applications can use this guide to better understand how to approach and complete a replatforming strategy to accelerate innovation, lower TCO, and increase developer productivity for their organization.

Contributors

Contributors to this document include:

- Andy Hopper, Principal Solutions Architect, Amazon Web Services
- Carlos Santos, Sr. Solutions Architect, Amazon Web Services
- Marcio Morales, Sr. Solutions Architect, Amazon Web Services
- Neeraj Handa, Sr. Solutions Architect, Amazon Web Services
- Chris Splinter, Sr. Product Manager, Amazon Web Services
- Yuvraj Mehta, Sr. Product Manager, Amazon Web Services

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Initial publication	Whitepaper first published.	January 4, 2022

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

Notices

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.