Red Hat OpenShift Service on AWS 4

ROSA architecture

An overview of the architecture for Amazon Managed Red Hat OpenShift 4
Red Hat OpenShift Service on AWS 4 ROSA architecture

An overview of the architecture for Amazon Managed Red Hat OpenShift 4
Abstract

This document provides an overview of the platform and application architecture in Amazon Managed Red Hat OpenShift 4.
# Table of Contents

## CHAPTER 1. INTRODUCTION TO ROSA  
1.1. ABOUT RED HAT OPENSHIFT SERVICE ON AWS  
   1.1.1. Cluster creation consoles  
   1.1.2. Consumption experience  
   1.1.3. Integration with AWS services  

## CHAPTER 2. ARCHITECTURE MODELS  
2.1. ROSA ARCHITECTURE ON PUBLIC AND PRIVATE NETWORKS  
2.2. PRIVATELINK ARCHITECTURE  
2.3. AWS REFERENCE ARCHITECTURES  

## CHAPTER 3. ARCHITECTURE CONCEPTS  
3.1. OPENSHIFT  
3.2. KUBERNETES  
3.3. CONTAINERS
CHAPTER 1. INTRODUCTION TO ROSA

Learn about Red Hat OpenShift Service on AWS (ROSA) access, supported consoles, consumption experience, and integration with Amazon Web Services (AWS) services.

1.1. ABOUT RED HAT OPENSIFT SERVICE ON AWS

Red Hat OpenShift Service on AWS (ROSA) is a fully-managed OpenShift service, jointly managed and supported by both Red Hat and Amazon Web Services (AWS). Having your clusters maintained by this service gives you the freedom to focus on deploying applications.

This service is licensed directly from your AWS account. ROSA pricing is consumption based and is billed directly to your AWS account.

You can quickly deploy OpenShift clusters directly from the AWS console and integrate with other AWS services.

1.1.1. Cluster creation consoles

To create a new cluster, start from the AWS Management console using ROSA. This integrates with the new rosa CLI and API to provision clusters in your AWS account. The new API for cluster creation alleviates the burden of manually deploying the cluster in your existing VPC and account.

1.1.2. Consumption experience

After the clusters are created, you can operate your clusters with the OpenShift web console or with the OpenShift Cluster Manager. The ROSA service also uses OpenShift APIs. These tools provide a standardized OpenShift experience to leverage your existing skills and tools knowledge.

You receive OpenShift updates with new feature releases and a shared, common source for alignment with OpenShift Container Platform. ROSA supports the same versions of OpenShift as Red Hat OpenShift Dedicated and OpenShift Container Platform to achieve version consistency.

1.1.3. Integration with AWS services

ROSA can integrate with a range of AWS compute, database, analytics, machine learning, networking, mobile, and various application services, enabling customers to benefit from the more than 170 AWS services that scale on-demand across the globe. These AWS-native services are directly accessible to quickly deploy and scale services through the same management interface.
CHAPTER 2. ARCHITECTURE MODELS

ROSA architecture supports the following network configuration types:

- Public network
- Private network
- AWS PrivateLink

2.1. ROSA ARCHITECTURE ON PUBLIC AND PRIVATE NETWORKS

You can install ROSA using either a public or private network. Configure a private cluster and private network connection during or after the cluster creation process. Red Hat manages the cluster with limited access through a public network. For more information, see the Service Definition.

Figure 2.1. ROSA deployed on public and private networks

Alternatively, install a cluster using AWS PrivateLink, which is hosted on private subnets only.

2.2. PRIVATELINK ARCHITECTURE

The Red Hat managed infrastructure that creates AWS PrivateLink clusters is hosted on private subnets. The connection between Red Hat and the customer-provided infrastructure is created through PrivateLink VPC endpoints.

The following diagram shows a multiple availability zone (Multi-AZ) PrivateLink cluster deployed on private subnets.

Figure 2.2. Multi-AZ PrivateLink cluster deployed on private subnets
2.3. AWS REFERENCE ARCHITECTURES

AWS provides multiple reference architectures that can be useful to customers when planning how to set up a configuration that uses PrivateLink. Here are three examples:

- **VPC with a private subnet and AWS Site-to-Site VPN access.**
  This configuration enables you to extend your network into the cloud without exposing your network to the internet.

  To enable communication with your network over an Internet Protocol Security (IPsec) VPN tunnel, this configuration contains a virtual private cloud (VPC) with a single private subnet and a virtual private gateway. Communication over the internet does not use an internet gateway.

  For more information, see [VPC with a private subnet only and AWS Site-to-Site VPN access](#) in the AWS documentation.

- **VPC with public and private subnets (NAT)**
  This configuration enables you to isolate your network so that the public subnet is reachable from the internet but the private subnet is not.

  Only the public subnet can send outbound traffic directly to the internet. The private subnet can access the internet by using a network address translation (NAT) gateway that resides in the public subnet. This allows database servers to connect to the internet for software updates using the NAT gateway, but does not allow connections to be made directly from the internet to the database servers.

  For more information, see [VPC with public and private subnets (NAT)](#) in the AWS documentation.

- **VPC with public and private subnets and AWS Site-to-Site VPN access**
  This configuration enables you to extend your network into the cloud and to directly access the internet from your VPC.

  You can run a multi-tiered application with a scalable web front end in a public subnet, and house your data in a private subnet that is connected to your network by an IPsec AWS Site-to-Site VPN connection.

  For more information, see [VPC with public and private subnets and AWS Site-to-Site VPN access](#) in the AWS documentation.
CHAPTER 3. ARCHITECTURE CONCEPTS

Learn about OpenShift and container basic concepts used in Red Hat OpenShift Service on AWS architecture.

3.1. OPENSHIFT

OpenShift is a Kubernetes container platform that provides a trusted environment to run enterprise workloads. It extends the Kubernetes platform with built-in software to enhance app lifecycle development, operations, and security. With OpenShift, you can consistently deploy your workloads across hybrid cloud providers and environments.

3.2. KUBERNETES

Red Hat OpenShift Service on AWS (ROSA) uses the Red Hat enterprise Kubernetes platform. Kubernetes is an open source platform for managing containerized workloads and services across multiple hosts, and offers management tools for deploying, automating, monitoring, and scaling containerized apps with minimal to no manual intervention. For complete information about Kubernetes, see the Kubernetes documentation.

Cluster, compute pool, and compute node

A Kubernetes cluster consists of a control plane and one or more compute nodes. Compute nodes are organized into compute pools of the type or profile of CPU, memory, operating system, attached disks, and other properties. The compute nodes correspond to the Kubernetes Node resource, and are managed by a Kubernetes control plane that centrally controls and monitors all Kubernetes resources in the cluster.

When you deploy the resources for a containerized app, the Kubernetes control plane decides which compute node to deploy those resources on, accounting for the deployment requirements and available capacity in the cluster. Kubernetes resources include services, deployments, and pods.

Namespace

Kubernetes namespaces are a way to divide your cluster resources into separate areas that you can deploy apps and restrict access to, such as if you want to share the cluster with multiple teams. For example, system resources that are configured for you are kept in separate namespaces like kube-system. If you do not designate a namespace when you create a Kubernetes resource, the resource is automatically created in the default namespace.

Pod

Every containerized app that is deployed into a cluster is deployed, run, and managed by a Kubernetes resource that is called a pod. Pods represent small deployable units in a Kubernetes cluster and are used to group the containers that must be treated as a single unit. In most cases, each container is deployed in its own pod. However, an app can require a container and other helper containers to be deployed into one pod so that those containers can be addressed by using the same private IP address.

App

An app can refer to a complete app or a component of an app. You can deploy components of an app in separate pods or separate compute nodes.

Service

A service is a Kubernetes resource that groups a set of pods and provides network connectivity to these pods without exposing the actual private IP address of each pod. You can use a service to make your app available within your cluster or to the public Internet.
A deployment is a Kubernetes resource where you can specify information about other resources or capabilities that are required to run your app, such as services, persistent storage, or annotations. You configure a deployment in a configuration YAML file, and then apply it to the cluster. The Kubernetes master configures the resources and deploys containers into pods on the compute nodes with available capacity.

Define update strategies for your app, including the number of pods that you want to add during a rolling update and the number of pods that can be unavailable at a time. When you perform a rolling update, the deployment checks whether the update is working and stops the rollout when failures are detected.

A deployment is just one type of workload controller that you can use to manage pods.

### 3.3. CONTAINERS

Containers provide a standard way to package your application code, configurations, and dependencies into a single unit. Containers run as isolated processes on compute hosts and share the host operating system and its hardware resources. A container can be moved between environments and run without changes. Unlike virtual machines, containers do not virtualize a device, its operating system, and the underlying hardware. Only the app code, run time, system tools, libraries, and settings are packaged inside the container. This approach makes a container more lightweight, portable, and efficient than a virtual machine.

Built on existing Linux container technology (LXC), the OCI-compliant container images define templates for how to package software into standardized units that include all of the elements that an app needs to run. Red Hat OpenShift Service on AWS (ROSA) uses CRI-O as the container runtime to deploy containers to your cluster.

To run your app in Kubernetes on ROSA, you must first containerize your app by creating a container image that you store in a container registry.

**Image**

A container image is the base for every container that you want to run. Container images are built from a Dockerfile, a text file that defines how to build the image and which build artifacts to include in it, such as the app, the app configuration, and its dependencies. Images are always built from other images, making them quick to configure.

**Registry**

An image registry is a place to store, retrieve, and share container images. Images that are stored in a registry can either be publicly available (public registry) or accessible by a small group of users (private registry). ROSA offers public images that you can use to create your first containerized app. For enterprise applications, use a private registry to protect your images from being used by unauthorized users.