Reference Architecture, Deployment Topology, and Value Proposition

Telco Edge Workloads on Red Hat OpenShift - Wavelength and Local Zones

Telco Edge Workloads on Red Hat OpenShift - Wavelength and Local Zones: Reference Architecture, Deployment Topology, and Value Proposition

Table of Contents

Telco Edge Workloads on Red Hat OpenShift - Wavelength and Local Zones

Publication date: **September 8, 2022** (*Document revisions*)

Abstract

Technology modernization is one of the core strategic initiatives by chief technology officers (CTOs) and chief information officers (CIOs) to apply innovative technologies. Telecom and media providers are disrupted every few years by market shifting innovations. To compete, they need to develop and monetize on new services before they lose their competitive advantage.

Telco companies need rapid tools that allow them to move to the cloud and embrace the cloud ecosystems. Container technologies have revolutionized the way infrastructure is deployed and have become the standard for modernizing the next generation of applications using a combination of agile, DevOps, and microservices methodologies.

Telco transformation requires a virtualized and containerized Telco cloud architecture to modernize legacy networks. Container technologies act as the key enabler to meet the needs of cloud native applications. This whitepaper outlines major use cases, provides a reference architecture, and outlines best practices of hybrid cloud strategy for deploying telecom edge workloads using Red Hat OpenShift on AWS, using AWS Wavelength and Local Zones.

Are You Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems.

Using the AWS Well-Architected Tool, available at no charge in the

AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

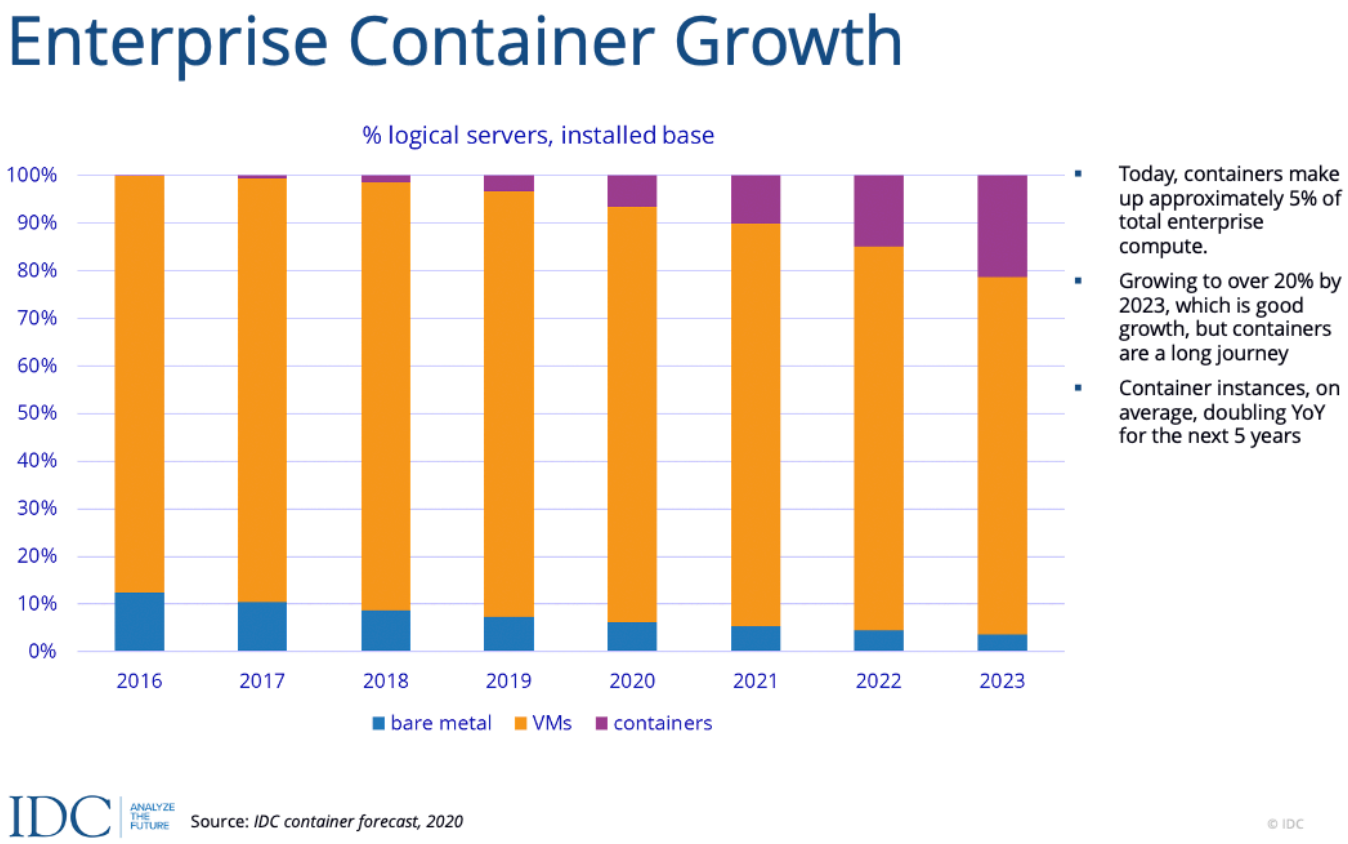
For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the

AWS Architecture Center.

Introduction

Containers are portable and highly efficient in encapsulating cloud native applications. They enable a broad range of other technologies, such as automation and orchestration, continuous integration/deployment (CI/CD), microservices, and immutable infrastructure. The International Data Corporation (IDC) forecasts that by 2023, there will be roughly 1.8 billion enterprise containers deployed, representing a five-year compound annual growth rate (CAGR) of 79%.

An effective cloud adoption program using practices guided by experience can lead organizations to successful container-based software delivery infrastructure. Communications service providers (CSPs) understand the necessity and importance of becoming cloud native, but don't know where to start. This whitepaper describes an approach for deploying Telco and Media Edge workloads as containerized applications using Red Hat OpenShift on AWS across AWS Wavelength and Local Zones.



IDC Report on Market Analysis Perspective: Worldwide Software-Defined Compute, 2020

IDC Report on Market Analysis Perspective: Worldwide Software-Defined Compute, 2020

Hybrid cloud

Hybrid cloud combines and unifies public cloud and private cloud services from cloud vendors to create a single, flexible, cost-optimized IT infrastructure. Many businesses and organizations now adopt cloud computing as a key aspect of their technology strategy. Businesses are moving their workloads to the AWS Cloud for greater agility, cost savings, performance, availability, resiliency, and scalability.

While most applications can be easily migrated, some applications need to be re-architected or modernized before they can be moved to the cloud. These applications must remain on-premises due to low-latency, local data processing, high data transfer costs, or data residency requirements. This leads many organizations to seek hybrid cloud architectures to integrate their on-premises and cloud operations to support a broad spectrum of use cases.

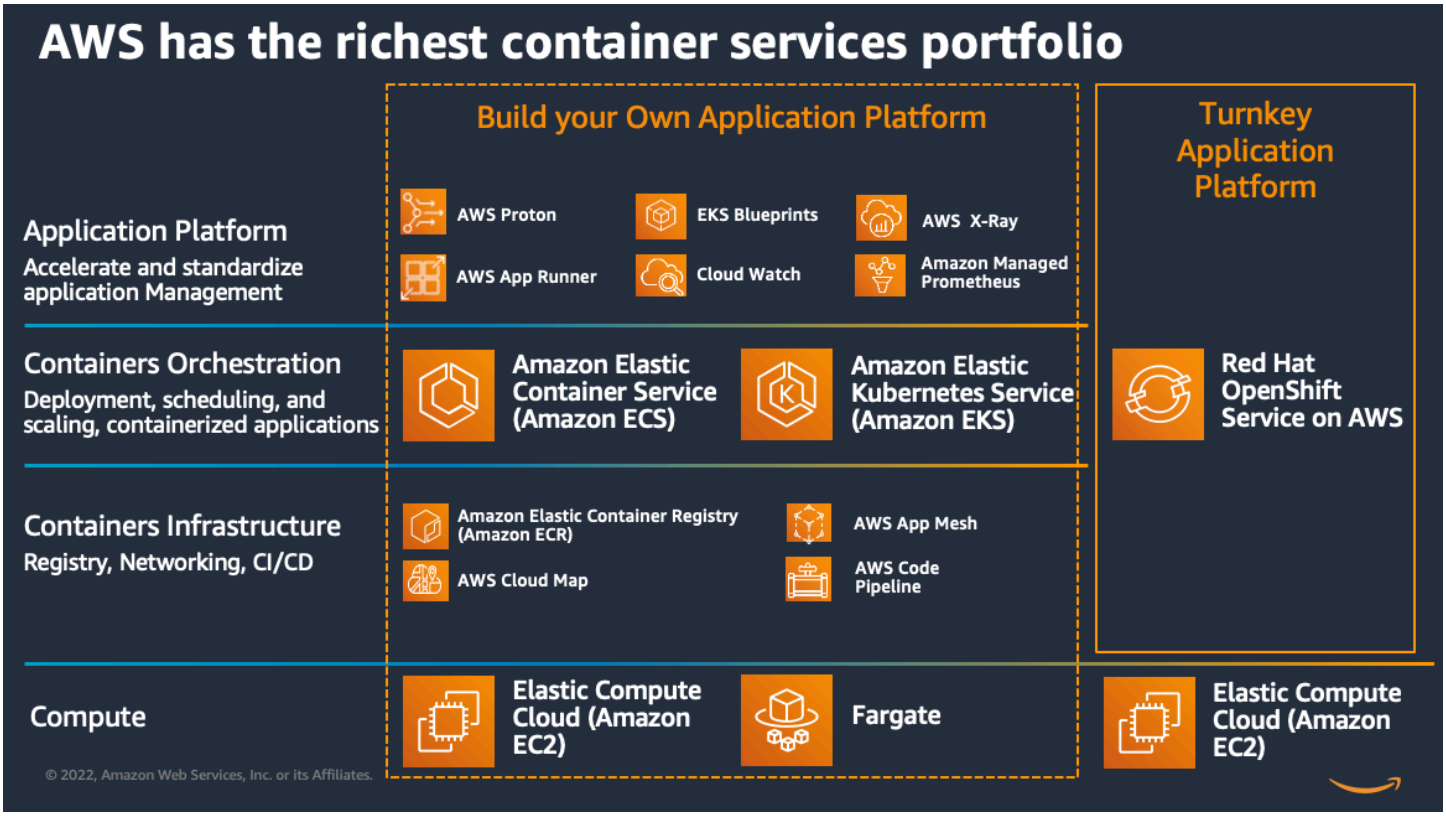
Open hybrid cloud strategies rely on a combination of on-premises hardware providers, leaning on multiple clouds for specific services. Red Hat OpenShift on AWS is designed to support IT organizations across the open hybrid cloud, regardless of the technical makeup. With accelerating 5G adoption by CSPs and the expansion of both private and public mobile edge computing (MEC), it’s imperative for CSPs to run their workload spread across regions and a combination of both on-premises infrastructure near-edge and far-edge zones.

AWS Wavelength embeds AWS compute and storage services within 5G networks, providing mobile edge computing infrastructure for developing, deploying, and scaling ultra-low-latency applications. For ultra-reliable low latency (URLL) use cases like real-time gaming and live streaming, augmented and virtual reality (AR/VR). In addition, local zones can place compute, storage, and AWS native services close to large metro zones and industry centers.

Container platforms

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 (VMs), containers don't 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 VM. AWS has the richest container services portfolio, with Amazon Elastic Kubernetes Service

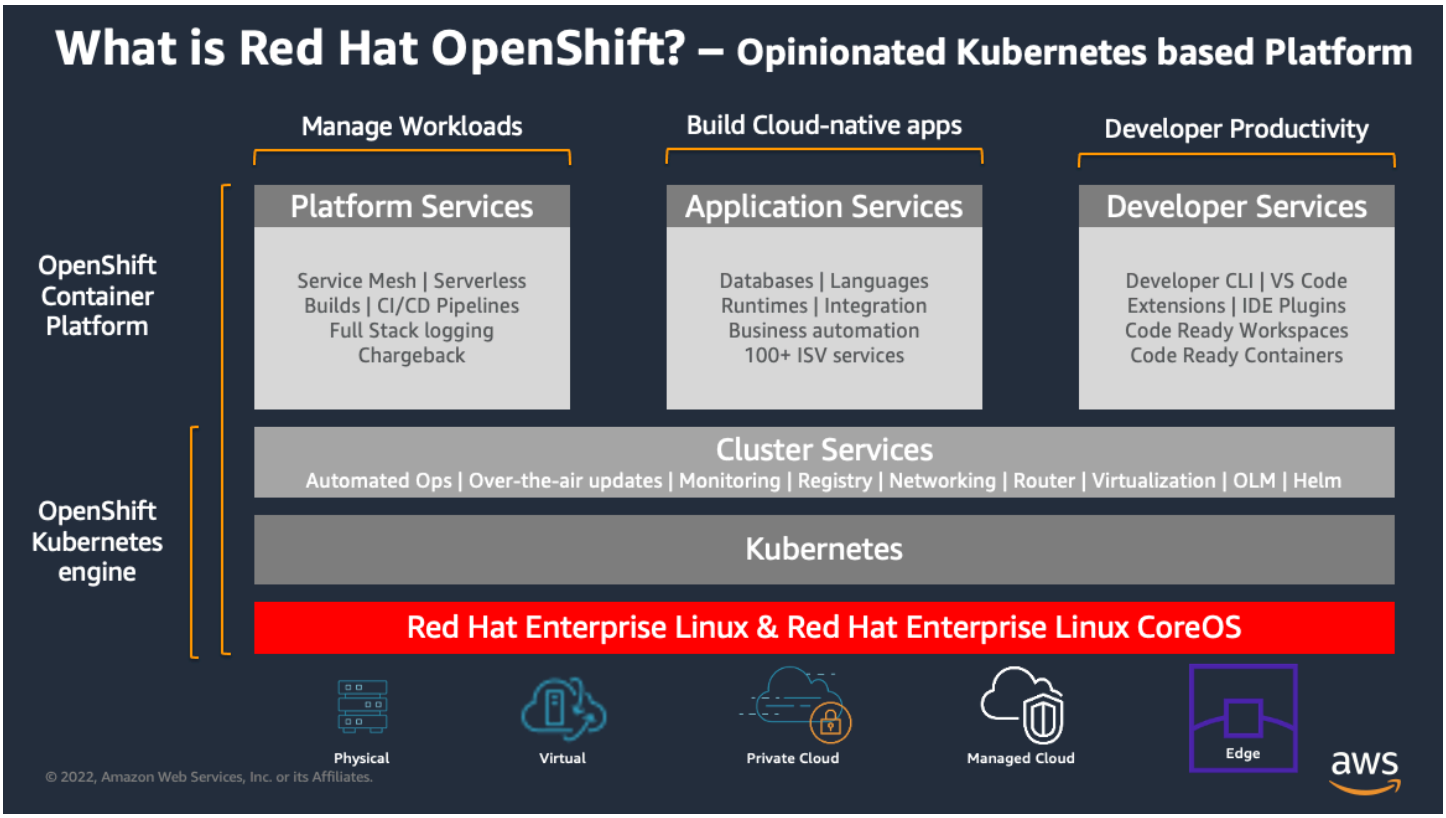
(Amazon EKS), Amazon Elastic Container Service (Amazon ECS), AWS Fargate (serverless Kubernetes) and Red Hat OpenShift Service on AWS (ROSA).



AWS container services portfolio

RedHat OpenShift Container Platform

Red Hat OpenShift Container Platform is a comprehensive enterprise-ready container solution built around Kubernetes. It includes both infrastructure and operations to enable the full stack developer experience. Red Hat OpenShift combined with AWS helps teams accelerate development and delivery of Kubernetes applications across a unified hybrid cloud environment. OpenShift helps organizations implement a Kubernetes infrastructure designed for rapid application development and deployment. By delivering more of the open-source projects you need along with Kubernetes, the OpenShift platform enables IT operations and developers to collaborate effectively and deploy containerized applications.



Red Hat OpenShift components

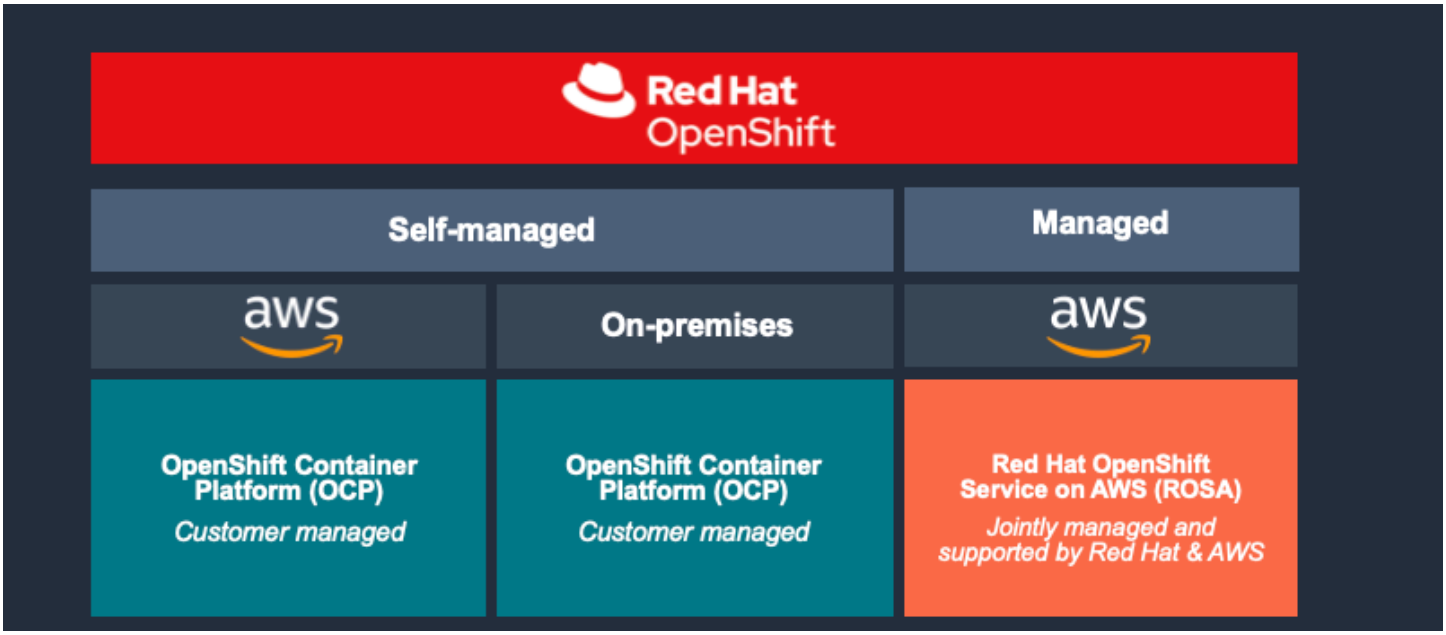
Red Hat OpenShift is designed to allow applications and the data centers that support them to expand from just a few machines and applications to thousands of machines that serve millions of clients. With its foundation in Kubernetes, OpenShift Container Platform incorporates the same technology that serves as the engine for massive telecommunications, streaming video, gaming, banking, and other applications. Kubernetes is an open-source container orchestration engine for automating deployment, scaling, and management of containerized applications.

OpenShift architecture builds on top of Kubernetes and is comprised of three types of nodes:

- **Control Plane Nodes** — Kubernetes Control Plane Nodes that might provide additional functionalities like the web console with self service capabilities
- **Infrastructure Nodes** — Kubernetes Worker nodes dedicated to host functionalities like routing and registry
- **App Nodes or Nodes** — Kubernetes worker nodes used to run the microservices and containerized applications deployed on OpenShift

Red Hat OpenShift on AWS

Red Hat OpenShift Service on AWS (ROSA) is a fully managed and jointly supported Red Hat OpenShift offering that combines the power of Red Hat OpenShift, the industry's most comprehensive enterprise Kubernetes platform, and the AWS public cloud. ROSA provides an integrated experience to use OpenShift. If you're already familiar with OpenShift, you can accelerate your application development process by leveraging familiar OpenShift APIs and tools for deployments on AWS. With ROSA, you can use the wide range of AWS compute, database, analytics, ML, networking, mobile, and other services to build secure and scalable applications faster. Red Hat OpenShift Service on AWS comes with pay-as-you-go hourly and annual billing, a 99.95% service-level agreement (SLA), and joint support from AWS and Red Hat.



Red Hat OpenShift deployment models

Benefits of Red Hat OpenShift on AWS:

- ROSA accelerates application development and testing lifecycles throughout the Enterprise IT architecture without being bounded by the limitations of framework, any deployment topology, or programming language inconsistencies.
- ROSA accelerates the adoption of DevOps so your development team can focus on designing and testing applications rather than spending time in managing and deploying containers.

- ROSA provides containerization for multitenancy, automatic provisioning, container security, monitoring, automatic application scaling, continuous integration, and self-service for developers.

ROSA architecture

Red Hat OpenShift Service on AWS is a managed service, available on the AWS Management Console, that makes it easier for Red Hat OpenShift customers to build, scale, and manage containerized applications on AWS. With ROSA, customers can quickly and easily create Kubernetes clusters using familiar Red Hat OpenShift APIs and tooling, and seamlessly have access to the full breadth and depth of AWS services. ROSA streamlines moving on-premises Red Hat OpenShift workloads to AWS, and offers a tighter integration with other AWS services. ROSA also enables customers to access Red Hat OpenShift licensing, billing, and support directly through AWS, delivering the simplicity of a single-vendor experience to customers.

Red Hat OpenShift on AWS private cluster architecture

Red Hat OpenShift Service on AWS 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.

Relevant Kubernetes resources include:

- **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 central processing unit (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 n*amespaces* are a way to divide your cluster resources into separate areas that you can deploy apps and restrict access to – for example, if you want to share the cluster with multiple teams, system resources that are configured for you are kept in separate namespaces like kube-system. If you don't 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 called a *pod*. Pods represent small deployable units in a Kubernetes cluster and are used to group the containers that you want 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** — *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.
- **Deployment** — 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 Control Plane configures the resources and deploys containers into pods on the compute nodes with available capacity. The control plane also defines update strategies for the hosted 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.

Value proposition of ROSA

Red Hat OpenShift on AWS offers a wide range of benefits for developers, IT operations and business leaders. Containers add a layer of abstraction that isn't present in VMs. While VMs rely on the infrastructure layer to provide benefits such as resilience, containers are cloud-native and are built to be independent of their infrastructure. This abstraction also enhances security, not only because patches can be rolled out faster, but also because just the container host can be patched – as opposed to multiple, individual guest operating systems that each need attention. By making application development faster, scaling easier, and management less complex, containers allow providers to launch newer applications and services faster and gain a competitive advantage.

Red Hat and AWS have collaborated to make it easy to run Red Hat Enterprise Linux on AWS since 2008, and we are expanding on that collaboration for Red Hat OpenShift on AWS. You can now acquire Red Hat OpenShift licensing through AWS, and then quickly deploy managed OpenShift clusters in your account. By working together, we're now able to provide ROSA with a set of features for the best OpenShift experience on AWS.

- **AWS Management Console integration and streamlined OpenShift cluster creation** — You can get started with the Red Hat OpenShift Service on AWS through the AWS Management Console, and a new CLI and API to provision clusters in your account. After you've created your clusters, you can manage them through the familiar OpenShift Console or with the OpenShift Cluster Manager.
- **Standard Red Hat OpenShift clusters consumption experience** — To move more quickly, customers find value in being able to use familiar skills and tooling. This new service has the same familiar OpenShift APIs, so you can lean on existing skills and tools for operating your clusters. Customers will continue to receive OpenShift updates with new feature releases and share a 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 everywhere. ROSA adds a new API for cluster creation to alleviate the burden of manually deploying the cluster in your existing VPC and account, without getting in the way of how you use it.
- **Out of the box integration with AWS infrastructure** — Developers can easily deploy applications with dependencies on AWS services by using the OpenShift Service Catalog and AWS Service Broker, an implementation of the Open Service Broker API. The AWS Service Broker provides an intermediate layer that allows users to deploy services using native manifests and the OpenShift Console. AWS Service Broker supports a subset of AWS services, including Amazon

Relational Database Service (Amazon RDS), Amazon EMR, Amazon DynamoDB, Amazon Simple Storage Service (Amazon S3), and Amazon Simple Queue Service (Amazon SQS).

- **Managed service experience provided by both AWS and Red Hat** — We want to help you avoid going through multi-page manuals to stand up a production-grade OpenShift cluster on AWS. Having your precious engineering resources spend cycles managing clusters for regular maintenance isn't the best way to keep them busy. Those engineering resources can (and should) be used to create value to the business instead.
- **Consumption-based pricing with no upfront costs** — Our customers tell us that a consumption-based model is one of the main reasons they moved to the cloud in the first place. Consumption-based pricing allows you to experiment and fail fast, and customers have told us they want to align their Red Hat OpenShift licensing consumption with how they plan to operate in AWS. As a result, we are providing an hourly pay-as-you-go model and annual commitments for customers who can take advantage of up-front commitments.
- **Integrated AWS billing experience** — While this is a service jointly managed and supported by Red Hat and AWS, you only have to deal with a bill from a single vendor: AWS. Each AWS service supporting your cluster components and application requirements is still a separate billing line item, but now with the addition of your OpenShift subscription. For example, all the infrastructure related components (instances, load balancers, storage) are reported as standard AWS line items, while the Red Hat OpenShift subscription is listed with other AWS Marketplace subscriptions. We think this is positive news for our joint AWS and Red Hat customers because they can now have a unified vendor experience for adoption and continue to build on their existing Red Hat relationship through AWS Marketplace private offers.

AWS Local Zones

A Local Zone is an extension of an AWS Region that is geographically close to your users. You can extend any virtual private cloud (VPC) from the parent AWS Region into Local Zones by creating a new subnet and assigning it to the AWS Local Zone. When you create a subnet in a Local Zone, your VPC is extended to that Local Zone. The subnet in the Local Zone operates the same as other subnets in your VPC. AWS Local Zones allow you to use select AWS services, like compute and storage services, closer to more end users, giving them low latency access to their local applications. AWS Local Zones are also connected to the parent region by using Amazon's redundant and high-bandwidth private network, giving applications running in AWS Local Zones fast, secure, and seamless access to the rest of AWS services.

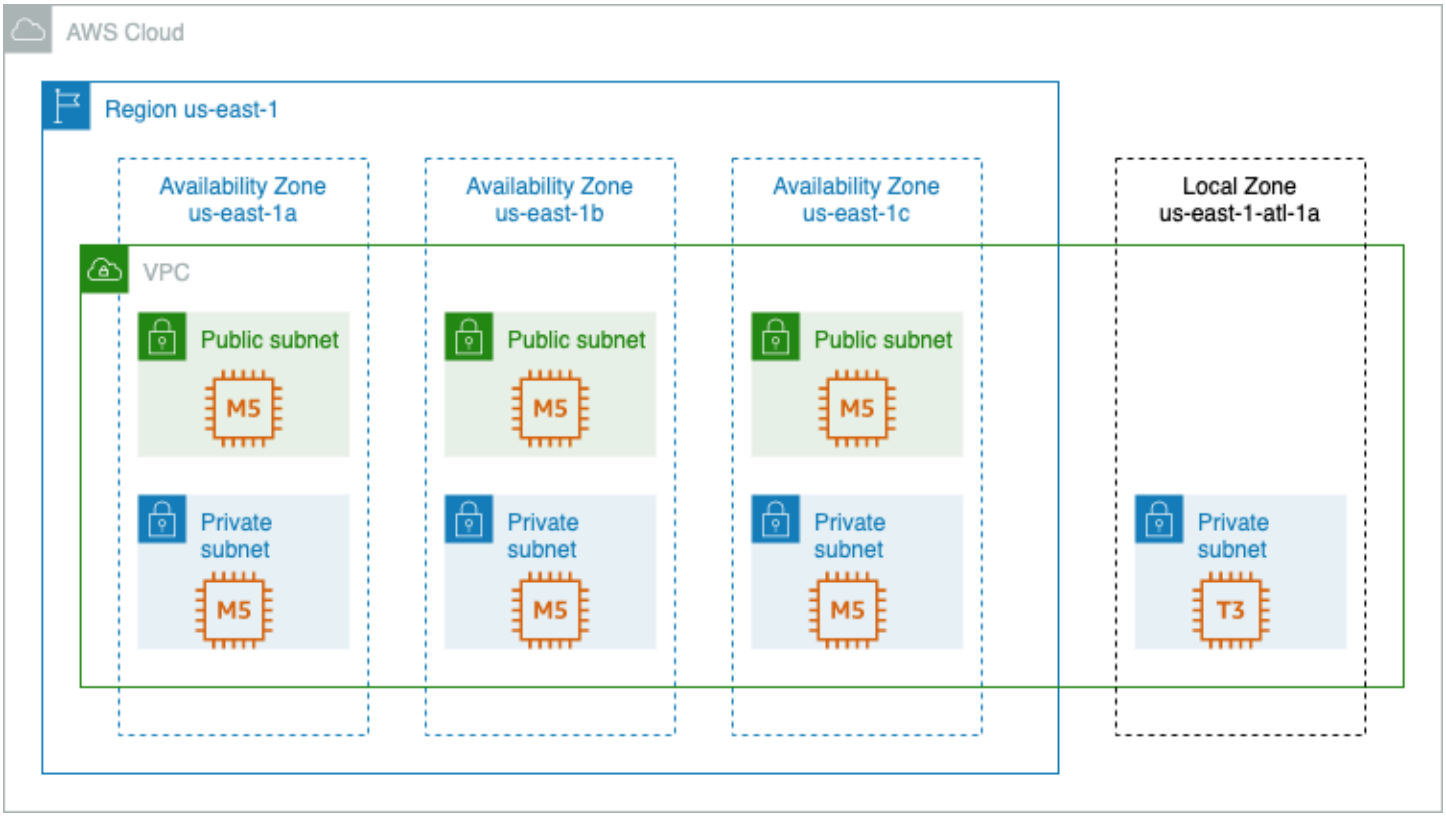
Red Hat OpenShift on AWS Local Zones

Reference architecture on AWS Local Zones

Local Zones are designed to bring the core services needed for the latency sensitive portions of your workload closer to end users, while Availability Zones provide access to the full array of AWS services. Services such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Block Store (Amazon EBS), Amazon VPC are locally available and can be used to serve end users in geographic proximity with extremely low latency, while other AWS services like Amazon S3 and Amazon Aurora are accessible privately in a VPC over an AWS private network.

Local Zones and Availability Zones help you build applications for high availability. AWS Local Zones are a type of AWS infrastructure deployment that places compute, storage, database, and other select services closer to large population, industry, and IT centers, enabling you to deliver applications that require single-digit millisecond latency to end users. The AWS Local Zones can run various AWS services such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Virtual Private Cloud (Amazon VPC), Amazon Elastic Block Store (Amazon EBS), Amazon FSx, Amazon Elastic Load Balancing, Amazon EMR, Amazon ElastiCache and Amazon Relational Database Service (Amazon RDS) in geographic proximity to your end users, with more services to be added in the future.

The following reference architecture illustrates the AWS Region `us-east-1`, two of its Availability Zones, and two of its Local Zones. The VPC spans the Availability Zones and one of the Local Zones. Each zone in the VPC has one subnet.



Red Hat OpenShift on AWS Local Zone architecture

Red Hat OpenShift on AWS Local Zones - Deployments

Deploy OpenShift worker nodes in AWS Local Zones to run latency-sensitive Kubernetes applications closer to end users to enable real-time gaming, live streaming, augmented and virtual reality (AR/VR), virtual workstations, and more. Another use case is to comply with state and local data residency requirements in sectors such as healthcare, financial services, gaming, and government.

We outline the deployment topology to deploy and manage a Red Hat OpenShift cluster in Region us-east-1 and then deploy a worker node in Local Zone Atlanta, `us-east-1-atl-1a`. We deploy Red Hat OpenShift using Installer-Provisioned-Infrastructure (IPI) on an existing VPC with predefined subnets. IPI simplifies the creation of VPC and the required networking constructs allocating the needed CIDR's for additional subnets in the Local Zone.

Create the VPC and subnets

There are multiple ways to create the VPC and the subnets. You can use Terraform, AWS CloudFormation, AWS CLI or AWS Management Console. We will use a CloudFormation template from [OpenShift documentation] with the following parameters to create the VPC and the subnets.

1. CloudFormation template:

```
vpc-parameters. json
[
{
"ParameterKey": "VpcCidr",
"ParameterValue": "10.0.0.0/16"
},
{
"ParameterKey": "AvailabilityZoneCount",
"ParameterValue": "3"
},
{
"ParameterKey": "SubnetBits",
"ParameterValue": "12"
}
]
```

2. Create the VPC and subnet using an AWS CLI command.

```
$ aws cloudformation create-stack --stack-name localzones-vpc --template-body file://localzones-vpc.yaml --parameters file://vpc-parameters.json –region us-east-1
```

3. Verify the VPC and the subnets are created and CF stack template is completed.

Your VPCs (2) Info

Filter VPCs

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR
–	vpc-0b9d709f3ea11b2c1	Available	172.31.0.0/16	–
lz-vpc	vpc-050f61b864b9b2ba9	Available	10.0.0.0/16	–

Your VPCs

Subnets (12) Info

Filter subnets

Name	Subnet ID	State	VPC	IPv4 CIDR
lz-public-subnet-1c	subnet-0e45490fa22d0f9e2	Available	vpc-050f61b864b9b2ba9 | lz-...	10.0.32.0/20
lz-private-subnet-1b	subnet-0bb71030f8d9c1c94	Available	vpc-050f61b864b9b2ba9 | lz-...	10.0.64.0/20
lz-private-subnet-1a	subnet-07ca600d2216d395c	Available	vpc-050f61b864b9b2ba9 | lz-...	10.0.48.0/20
lz-public-subnet-1b	subnet-0e3f0599be5e5bec1	Available	vpc-050f61b864b9b2ba9 | lz-...	10.0.16.0/20
lz-private-subnet-1c	subnet-088b6cb411f8b0d5e	Available	vpc-050f61b864b9b2ba9 | lz-...	10.0.80.0/20
lz-public-subnet-1a	subnet-0be63e6b666d7d3c1	Available	vpc-050f61b864b9b2ba9 | lz-...	10.0.0.0/20

Your subnets

4. Deploy OpenShift using IPI on the VPC you have just created. Here is the `install-config.yaml` sample configuration:

```
baseDomain: ocp.ovsandbox.com
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  platform: {}
  replicas: 2
```

```
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform: {}
  replicas: 3
metadata:
  creationTimestamp: null
  name: lz
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
  - cidr: 10.0.0.0/16
  networkType: OVNKubernetes
  serviceNetwork:
  - 172.30.0.0/16
platform:
  aws:
    region: us-east-1
    subnets:
    - subnet-0e45490fa22d0f9e2
    - subnet-0bb71030f8d9c1c94
    - subnet-07ca600d2216d395c
    - subnet-0e3f0599be5e5bec1
    - subnet-088b6cb411f8b0d5e
    - subnet-0be63e6b666d7d3c1
publish: External
pullSecret: '{"auths":...}'
sshKey: |
  ecdsa-sha2-nistp256 AAA...
```

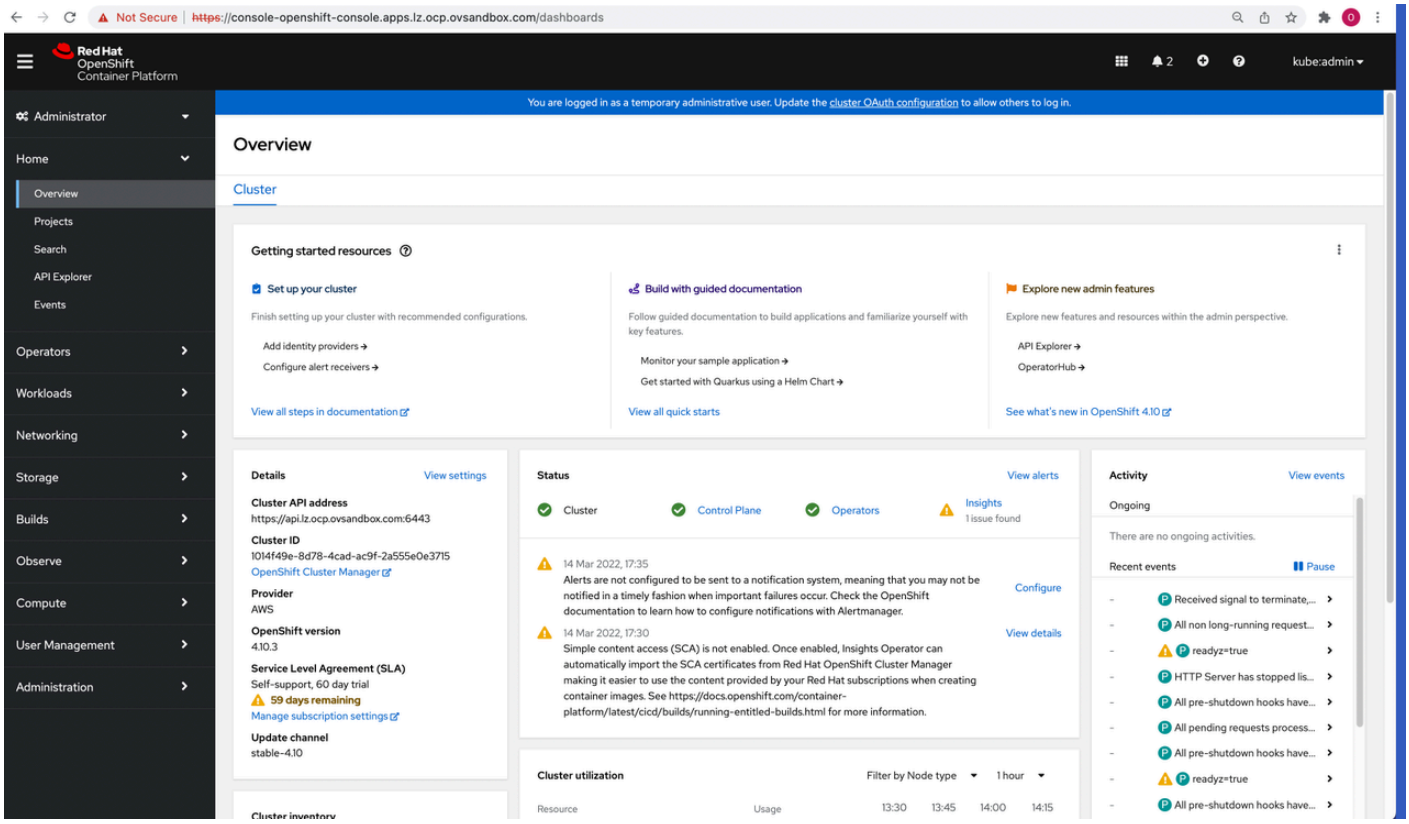
5. Create the OS cluster.

```
$ openshift-install create cluster –log-level=debug
```

It takes approximately 30 min to create the cluster.

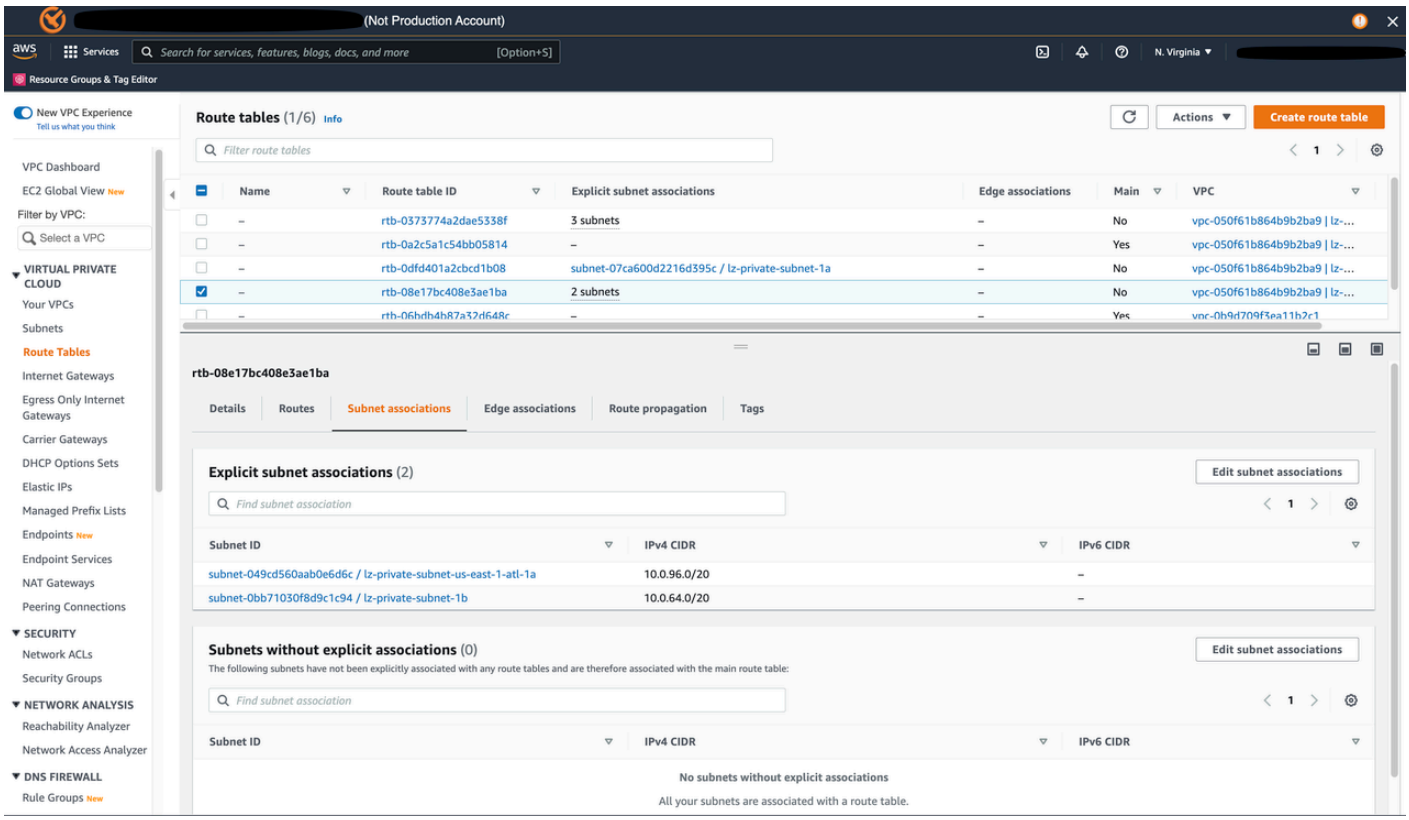
```
        INFO Install complete!
        INFO To access the cluster as the system:admin user when using 'oc', run
 'export KUBECONFIG=/home/ec2-user/ocp4/auth/kubeconfig'
        INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.lz.ocp.ovsandbox.com
        INFO Login to the console with user: "kubeadmin", and password: ""
        INFO Time elapsed: 30m45s
```

6. Once the installation is complete, log in to the OpenShift web console to check if the cluster is ready and operational.



Verify the OS cluster

7. Create the subnet in the cluster's VPC for AWS Local Zone. In the AWS Management Console, choose **Services > VPC**, then select Subnets. Create the subnet by providing the VPC ID, subnet name, Local Zone for Availability Zone, and CIDR.
8. Associate the subnets and NATs from the AWS Local Zone to a route table from one of the private subnets.



Subnet associations

A route to an existing NAT Gateway is added to the Local Zone private subnet.

9. Create a MachineSet for Local Zones. Verify what types of instances are available in the Local Zone. In this deployment topology, we are using `us-east-1-atl-1a`.

```
        $ aws ec2 describe-instance-type-offerings --location-type "availability-
zone" --filters Name=location,Values=us-east-1-atl-1a --region us-east-1 –output
 table
        | DescribeInstanceTypeOfferings |
        +-------------------------------------------------------------+
        || InstanceTypeOfferings ||
        |+--------------+--------------------+---------------------+|
        || InstanceType | Location | LocationType ||
        |+--------------+--------------------+---------------------+|
        || c5d.2xlarge | us-east-1-atl-1a | availability-zone ||
        || t3.xlarge | us-east-1-atl-1a | availability-zone ||
        || t3.medium | us-east-1-atl-1a | availability-zone ||
        || r5d.2xlarge | us-east-1-atl-1a | availability-zone ||
        || g4dn.2xlarge| us-east-1-atl-1a | availability-zone ||
```

```
          |+--------------+-------------------+--------------------+|
```

We use a t3.xlarge instance type for the worker node. This instance type is supported in OpenShift for a worker node. Next, create a MachineSet for the AWS Local Zones using t3.xlarge. MachineSet has a template for machine specifications.

```
          $ oc get machinesets -n openshift-machine-api
          NAME DESIRED CURRENT READY AVAILABLE AGE
          lz-d8lbp-worker-us-east-1a 1 1 1 1 20h
          lz-d8lbp-worker-us-east-1b 0 0 20h
          lz-d8lbp-worker-us-east-1c 1 1 1 1 20h
```

10. Use an existing MachineSet template and modify it for the Local Zones.

```
          $ oc get machineset lz-d8lbp-worker-us-east-1a -n openshift-machine-api -
oyaml > lz-machineset.yaml
```

In the file, replace all instances of `us-east-1a` with `us-east-1-atl-1a`, instanceType with t3.xlarge, and subnet id with the id of the subnet you created in Local Zones. The YAML file should look like this:

```
          apiVersion: machine.openshift.io/v1beta1
          kind: MachineSet
          metadata:
            labels:
              machine.openshift.io/cluster-api-cluster: lz-d8lbp
            name: lz-d8lbp-worker-us-east-1-atl-1a
            namespace: openshift-machine-api
          spec:
            replicas: 1
            selector:
              matchLabels:
                machine.openshift.io/cluster-api-cluster: lz-d8lbp
                machine.openshift.io/cluster-api-machineset: lz-d8lbp-worker-us-
east-1-atl-1a
```

```
            template:
              metadata:
                labels:
                  machine.openshift.io/cluster-api-cluster: lz-d8lbp
                  machine.openshift.io/cluster-api-machine-role: worker
                  machine.openshift.io/cluster-api-machine-type: worker
                  machine.openshift.io/cluster-api-machineset: lz-d8lbp-worker-us-
east-1-atl-1a
              spec:
                lifecycleHooks: {}
                metadata: {}
                providerSpec:
                  value:
                    ami:
                      id: ami-0efc96a4e17e7b048
                    apiVersion: awsproviderconfig.openshift.io/v1beta1
                    blockDevices:
                    - ebs:
                        encrypted: true
                        iops: 0
                        kmsKey:
                          arn: ""
                        volumeSize: 120
                        volumeType: gp3
                    credentialsSecret:
                      name: aws-cloud-credentials
                    deviceIndex: 0
                    iamInstanceProfile:
                      id: lz-d8lbp-worker-profile
                    instanceType: t3.xlarge
                    kind: AWSMachineProviderConfig
                    metadata:
                      creationTimestamp: null
                    placement:
                      availabilityZone: us-east-1-atl-1a
                      region: us-east-1
                    securityGroups:
                    - filters:
                      - name: tag:Name
                        values:
                        - lz-d8lbp-worker-sg
                    subnet:
                      id: subnet-049cd560localzone
```

```
                    tags:
                    - name: kubernetes.io/cluster/lz-d8lbp
                      value: owned
                    userDataSecret:
                      name: worker-user-data
```

11. Apply the MachineSet Manifest.

```
$ oc apply -f lz-machineset.yaml
```

12. Verify the new MachineSets.

```
$ oc get machineset -n openshift-machine-api
NAME DESIRED CURRENT READY AVAILABLE AGE
lz-d8lbp-worker-us-east-1-atl-1a 1 1 1 1 15m
lz-d8lbp-worker-us-east-1a 1 1 1 1 21h
lz-d8lbp-worker-us-east-1b 0 0 21h
lz-d8lbp-worker-us-east-1c 1 1 1 1 21h
```

13. Create a new machine and new OpenShift worker nodes.

```
$ oc get machines -n openshift-machine-api
NAME PHASE TYPE REGION ZONE AGE
lz-d8lbp-master-0 Running m6i.xlarge us-east-1 us-east-1b 21h
lz-d8lbp-master-1 Running m6i.xlarge us-east-1 us-east-1a 21h
lz-d8lbp-master-2 Running m6i.xlarge us-east-1 us-east-1c 21h
lz-d8lbp-worker-us-east-1-atl-1a-l5vk6 Running t3.xlarge us-east-1 us-
east-1-atl-1a 13m
lz-d8lbp-worker-us-east-1a-5v8hf Running m6i.large us-east-1 us-east-1a 21h
lz-d8lbp-worker-us-east-1c-jlhxj Running m6i.large us-east-1 us-east-1c 21h
```

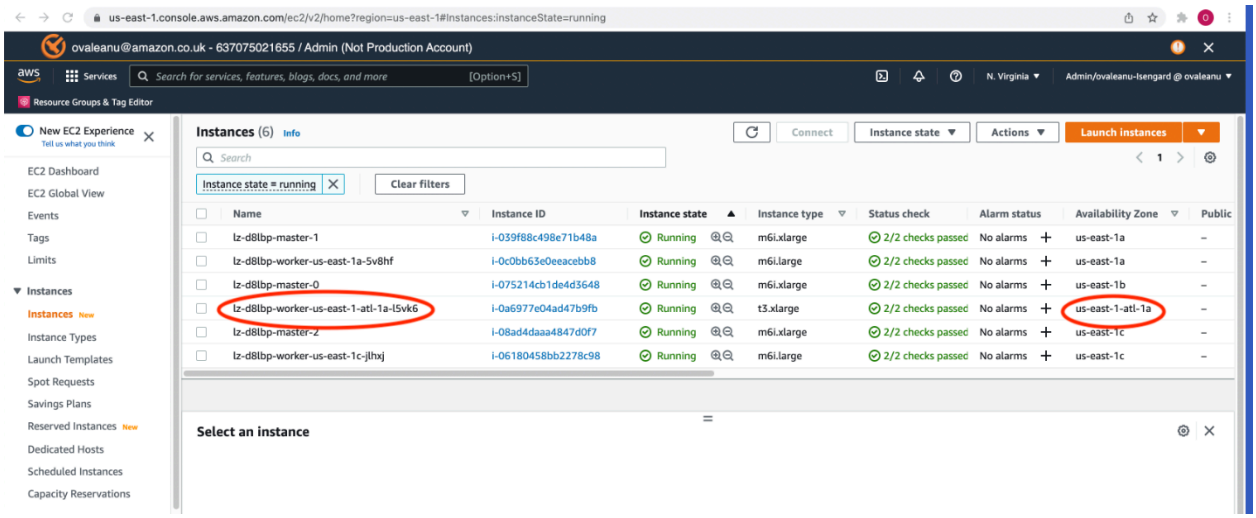
14. Verify the OpenShift worker nodes.

```
NAME STATUS ROLES AGE VERSION
ip-10-0-109-180.ec2.internal Ready worker 10m v1.23.3+e419edf
```

```
ip-10-0-49-112.ec2.internal Ready worker 21h v1.23.3+e419edf
ip-10-0-61-95.ec2.internal Ready master 21h v1.23.3+e419edf
ip-10-0-75-249.ec2.internal Ready master 21h v1.23.3+e419edf
ip-10-0-86-163.ec2.internal Ready worker 21h v1.23.3+e419edf
ip-10-0-86-234.ec2.internal Ready master 21h v1.23.3+e419edf
```

15. Verify OpenShift nodes at the Local Zone. Verify the new instance running at the Local Zone in AWS Management Console.

```
$ oc get nodes
NAME STATUS ROLES AGE VERSION
ip-10-0-109-180.ec2.internal Ready worker 10m v1.23.3+e419edf
ip-10-0-49-112.ec2.internal Ready worker 21h v1.23.3+e419edf
ip-10-0-61-95.ec2.internal Ready master 21h v1.23.3+e419edf
ip-10-0-75-249.ec2.internal Ready master 21h v1.23.3+e419edf
ip-10-0-86-163.ec2.internal Ready worker 21h v1.23.3+e419edf
ip-10-0-86-234.ec2.internal Ready master 21h v1.23.3+e419edf
```



Local Zone node verification

16. Deploy the OSToy sample application. The OSToy sample application is a two-tier application with the backend running on a worker node in the region and the frontend running on the node from the local zone. Create a new project for the OSToy deployment:

```
$ oc new-project ostoy
```

17 Label the worker nodes running in the region and local zone. Use nodeSelector, the simplest recommended form of node selection constraint. Label the worker node running on availability zone `us-east-1a`.

```
$ oc label nodes ip-10-0-49-112.ec2.internal availabilityZone=us-east-1a
```

Label the worker node running on availability zone (local zone) `us-east-1-atl-1a`.

```
. $ oc label nodes ip-10-0-109-180.ec2.internal availabilityZone=us-east-1-
atl-1a
```

18 Download the backend microservice deployment file.

```
$ wget https://raw.githubusercontent.com/openshift-cs/rosaworkshop/master/
rosa-workshop/ostoy/yaml/ostoy-microservice-deployment.yaml
```

Modify the downloaded file, ostoy-microservice-deployment.yaml, by adding a nodeSelector field the deployment configuration with the label you created for the application to run on the node running on availability zone us-east-1a.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ostoy-microservice
  labels:
    app: ostoy
spec:
    selector:
      matchLabels:
        app: ostoy-microservice
    replicas: 1
```

```
    template:
      metadata:
        labels:
          app: ostoy-microservice
      spec:
        containers:
        - name: ostoy-microservice
          image: quay.io/ostoylab/ostoy-microservice:1.4.0
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 8080
            protocol: TCP
          resources:
            requests:
              memory: "128Mi"
              cpu: "50m"
            limits:
              memory: "256Mi"
              cpu: "100m"
        nodeSelector:
          availabilityZone: us-east-1a

---
apiVersion: v1
kind: Service
metadata:
  name: ostoy-microservice-svc
  labels:
    app: ostoy-microservice
spec:
  type: ClusterIP
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  selector:
    app: ostoy-microservice
```

19Deploy the backend microservice using the deployment file.

```
$ oc apply -f ostoy-microservice-deployment.yaml
```

```
$ oc get po -owide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
ostoy-microservice-b74b4cc96-9stgc 1/1 Running 0 7s 10.128.2.102
ip-10-0-49-112.ec2.internal <none> <none>
```

The backend microservice is running on node `ip-10-0-49-112.ec2.internal` from availability zone `us-east-1a`.

20. Deploy the frontend services file and verify the created objects. Download the frontend microservice deployment file:

```
$ wget https://raw.githubusercontent.com/openshift-cs/rosaworkshop/master/
rosa-workshop/ostoy/yaml/ostoy-fe-deployment.yaml
```

Modify the downloaded file, ostoy-fe-deployment.yaml, by adding a nodeSelector field the deployment configuration with the label you created for the application to run on the node running on availability zone `us-east-1-atl-1a`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ostoy-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ostoy-frontend
  labels:
    app: ostoy
spec:
    selector:
      matchLabels:
```

```
      app: ostoy-frontend
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
        app: ostoy-frontend
    spec:
      containers:
      - name: ostoy-frontend
        image: quay.io/ostoylab/ostoy-frontend:1.4.0
        imagePullPolicy: IfNotPresent
        ports:
        - name: ostoy-port
          containerPort: 8080
        resources:
          requests:
            memory: "256Mi"
            cpu: "100m"
          limits:
            memory: "512Mi"
            cpu: "200m"
        volumeMounts:
        - name: configvol
          mountPath: /var/config
        - name: secretvol
          mountPath: /var/secret
        - name: datavol
          mountPath: /var/demo_files
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 10
          periodSeconds: 5
        env:
        - name: ENV_TOY_SECRET
          valueFrom:
            secretKeyRef:
              name: ostoy-secret-env
              key: ENV_TOY_SECRET
        - name: MICROSERVICE_NAME
```

```
                      value: OSTOY_MICROSERVICE_SVC
                  volumes:
                    - name: configvol
                      configMap:
                        name: ostoy-configmap-files
                    - name: secretvol
                      secret:
                        defaultMode: 420
                        secretName: ostoy-secret
                    - name: datavol
                      persistentVolumeClaim:
                        claimName: ostoy-pvc
                  nodeSelector:
                    availabilityZone: us-east-1-atl-1a

          ---
          apiVersion: v1
          kind: Service
          metadata:
            name: ostoy-frontend-svc
            labels:
              app: ostoy-frontend
          spec:
            type: ClusterIP
            ports:
              - port: 8080
                targetPort: ostoy-port
                protocol: TCP
                name: ostoy
            selector:
              app: ostoy-frontend
          ---
          apiVersion: v1
          kind: Route
          metadata:
            name: ostoy-route
          spec:
            to:
              kind: Service
              name: ostoy-frontend-svc
          ---
          apiVersion: v1
          kind: Secret
```

```
          metadata:
            name: ostoy-secret-env
          type: Opaque
          data:
            ENV_TOY_SECRET: VGhpcyBpcyBhIHRlc3Q=
          ---
          kind: ConfigMap
          apiVersion: v1
          metadata:
            name: ostoy-configmap-files
          data:
            config.json:  '{ "default": "123" }'
          ---
          apiVersion: v1
          kind: Secret
          metadata:
            name: ostoy-secret
          data:
            secret.txt:
VVNFUk5BTUU9bXlfdXNlcgpQQVNTV09SRD1AT3RCbCVYQXAhIzYzMlk1RndDQE1UUWsKU01UUD1sb2NhbGhvc3QKU01U
          type: Opaque

          $ oc apply -f ostoy-fe-deployment.yaml
          W0315 18:23:16.888918 1906 shim_kubectl.go:58] Using non-groupfied API
resources is deprecated and will be removed in a future release, update apiVersion
to "route.openshift.io/v1" for your resource
          persistentvolumeclaim/ostoy-pvc created
          deployment.apps/ostoy-frontend created
          service/ostoy-frontend-svc created
          route.route.openshift.io/ostoy-route created
          secret/ostoy-secret-env created
          configmap/ostoy-configmap-files created
          secret/ostoy-secret created
```

21. Verify frontend application deployment. The frontend app is running on the node from local zone.

```
          $ oc get po -owide
          NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
          ostoy-frontend-8cbfcfd86-m4cjx 1/1 Running 0 53s 10.129.2.24
ip-10-0-109-180.ec2.internal <none> <none>
```

```
        ostoy-microservice-b74b4cc96-9stgc 1/1 Running 0 6m41s 10.128.2.102
ip-10-0-49-112.ec2.internal <none> <none>
```

22. Get the route to the deployed application.

```
        $ oc get route
        NAME HOST/PORT PATH SERVICES PORT TERMINATION WILDCARD
        ostoy-route ostoy-route-ostoy.apps.lz.ocp.ovsandbox.com ostoy-frontend-svc
<all> None
```

Note

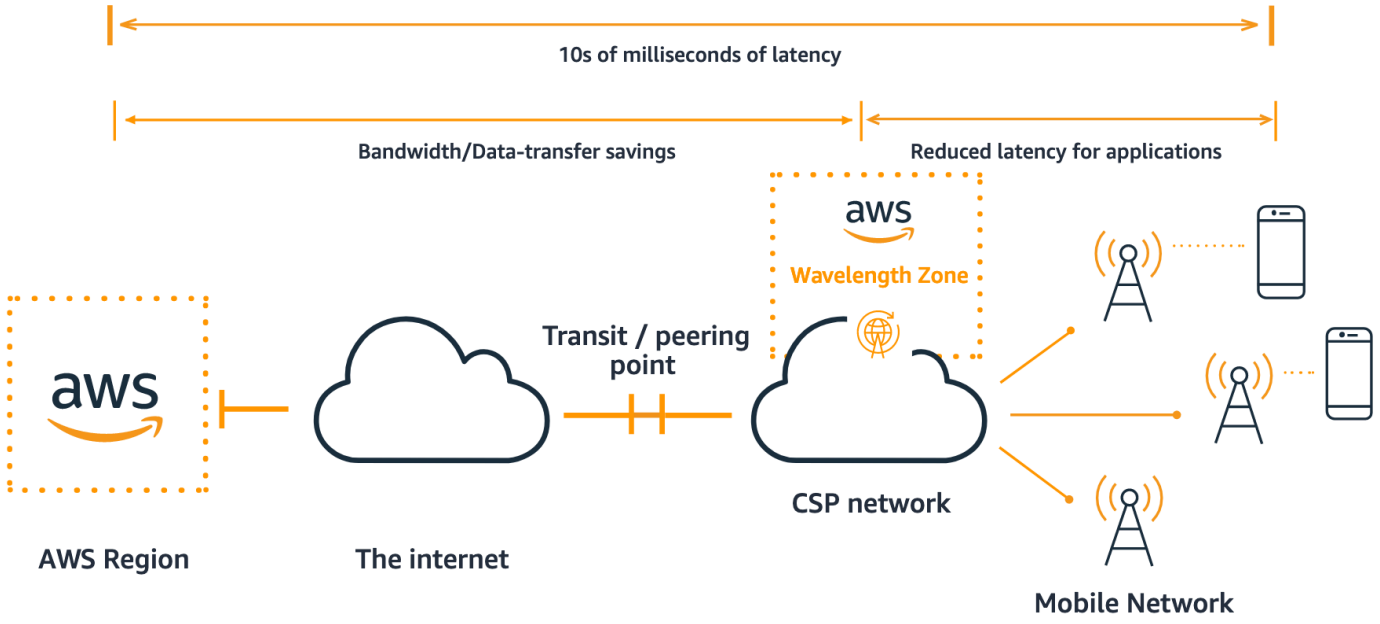
At the time of this writing, Network Load Balancer (Network Load Balancer) is not supported in AWS Local Zones; only Application Load Balancer (ALB) is supported. Red Hat OpenShift currently supports only Network Load Balancer. However, ALB can be configured as Ingress for the cluster. For the required steps to configure ALB, refer to Installing the AWS Load Balancer Controller (ALB) on ROSA.

AWS Wavelength Zones

Advances in radio technology have enabled 5G networks to provide high-density radio (air) interfaces with extremely high bandwidth and reliability. However, improvements in the radio network alone might not be enough to meet the low latency requirements set by the 5G standards. Today, most consumer and enterprise applications that are accessed on mobile devices and other mobile endpoints are hosted on application servers outside of the communications service provider's network.

Enabling applications to be run in edge computing infrastructure, close to end users, is essential to improving application latency. By running an application closer to its end point, the latency that comes from the number of hops needed for an application to reach the compute, storage, and cloud services it requires can be reduced. Accessing these resources in the cloud using traditional mobile architectures requires several hops on the network (from a device, to a cell tower, to metro aggregation sites, to regional aggregation sites, to the internet, to the cloud—and then back through those stops before getting back to the device). This creates tens to hundreds of milliseconds of latency.

The 5G network is up to ten times faster than 4G, but to take full advantage of the latency improvements that 5G offers, the number of network hops needs to be reduced.



AWS Wavelength for URRL and edge workloads

Reference architecture on AWS Wavelength Zones

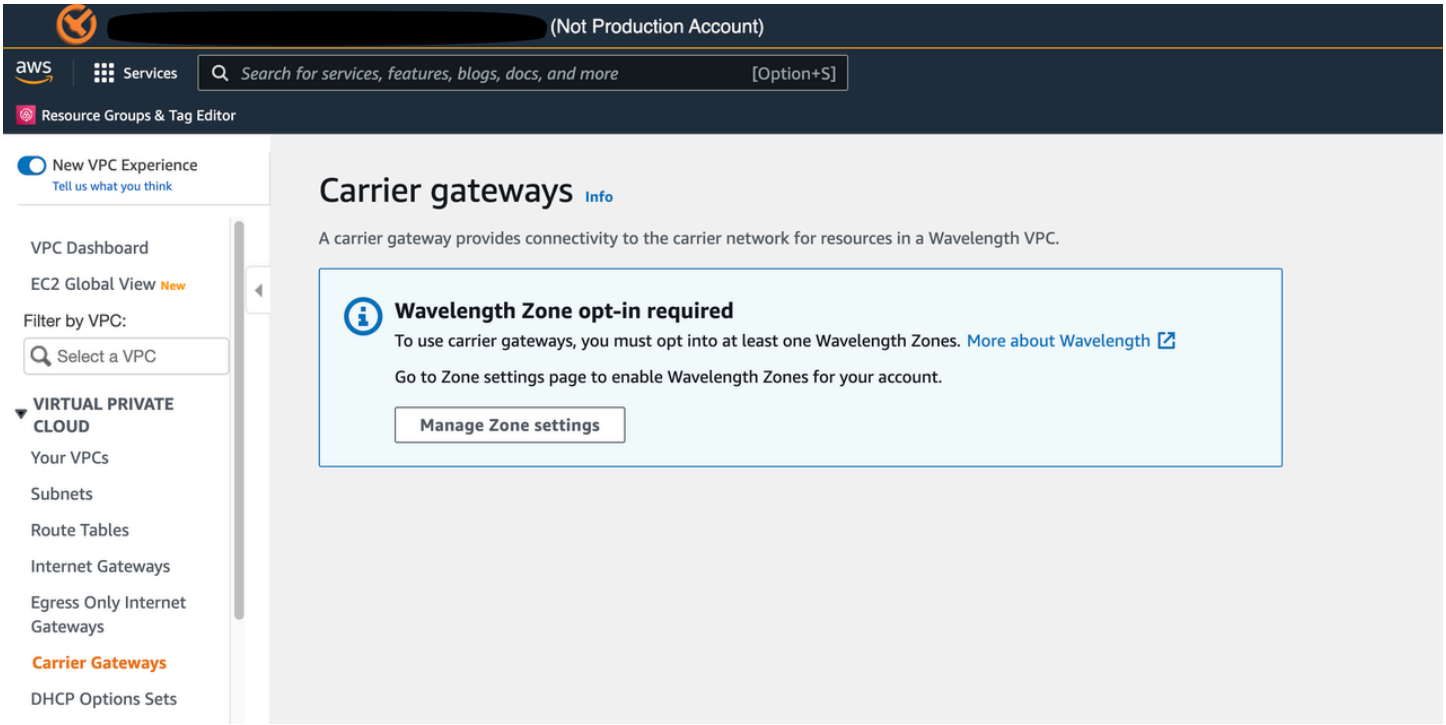
AWS Local Zones are an extension of an AWS Region, and provide the ability to place resources closer to the end users. In comparison, AWS Wavelength Zones allow developers to build applications that deliver ultra-low latencies to 5G devices and end users. Wavelength deploys standard AWS compute and storage services to the edge of telecommunication carriers' 5G networks. From the technical architectural perspective, the approach is similar to AWS Local Zones. The Amazon VPC can be extended to one or more Wavelength Zones and then use AWS resources like Amazon EC2 instances to run applications that require ultra-low latency and a connection to AWS services in the Region.

A Wavelength Zone is an isolated zone in the carrier location where the Wavelength infrastructure is deployed; they're tied to an AWS Region. A Wavelength Zone is a logical extension of a Region, and is managed by the control plane in the Region. A Wavelength Zone is represented by a region code followed by an identifier that indicates the Wavelength Zone (for example, `us-east-1-wl1-bos-wlz-1`).

To use a Wavelength Zone, you must opt in to the zone. After you opt in, create an Amazon VPC and subnet in the Wavelength Zone. Because deploying an OpenShift worker node on AWS Wavelength Zones is similar to doing the same in AWS Local Zones, only the steps specific to this setup are included.

Create a subnet in VPC for AWS Wavelength Zones

From the VPC where the OpenShift cluster is running, create a subnet for AWS Wavelength by providing the VPC ID, subnet name, Availability Zone (select Wavelength Zone) and CIDR. After the subnet for AWS Wavelength is created, create a Carrier gateway.



Opt in to the Wavelength Zone

Create a MachineSet for Wavelength Zones

Identically with how you create a MachineSet for Local Zones, create a MachineSet for Wavelength Zones. Replace the local zone and subnet ID with wavelength ID, and the subnet ID for it. Then apply the wavelength MachineSet to provision OpenShift worker nodes on the AWS Wavelength Zones.

Potential use cases

With global infrastructure that spans 77 Availability Zones in 24 AWS Regions, AWS enables developers to serve end users with low latencies worldwide. The following use cases – interactive applications, game streaming, virtual reality, near real-time rendering, industrial automation, smart cities, IoT, and autonomous vehicles – can benefit from the architectures outlined in this whitepaper.

- **Connected vehicles** — Cellular Vehicles to Everything (C-V2X) is an increasingly important platform for enabling intelligent driving, real-time HD maps, road safety, and more. Low latency access to compute infrastructure needed to run data processing and analytics on AWS Wavelength and Local Zones enables real-time monitoring of data from sensors for secure connectivity, in-car telematics, and autonomous driving.
- **Real-time gaming** — Real-time game streaming depends on low latency to preserve the user experience. With AWS Wavelength, the most demanding games can be made available on end-user devices that have limited processing power by streaming these games from game servers in Wavelength Zones.
- **Interactive live video streams** — Wavelength provides the ultra-low latency needed to livestream high-resolution video and high-fidelity audio, as well as to embed interactive experiences into live video streams. Additionally, real-time video analytics provide the ability to generate real-time stats that can enhance live event experiences.
- **Smart factories** — Industrial automation applications use machine learning (ML) inference at the edge to analyze images and videos in order to detect quality issues on fast-moving assembly lines and trigger actions to remediate the problem.

Advantages of running telco edge workloads on Red Hat OpenShift in AWS

- Edge computing coupled with the 5G network enables new classes of cloud applications in areas such as industrial robotic and drone automation, connected vehicles, and AR/VR infotainment. Innovations in business models will follow. Edge computing is essential for many emerging applications, which need local information processing to reduce the volume of traffic transported back to centralized datacenters. By enabling compute capabilities closer to end users, developers

and enterprises can provide innovative 5G applications and deliver immersive experiences to a wide audience.

- Time-to-market is accelerated by allowing CSPs to quickly deploy new edge services that leverage Amazon's compute, network, and storage capabilities. CSPs can achieve more agility because they do not need to install specialized new hardware to support changing business requirements. Edge deployments allow CSPs to develop services to meet the growing Industry 4.0 demands, thus reducing the risk associated with new services.
- It delivers agility and flexibility by allowing you to quickly scale services to address changing demands, and supports innovation by enabling service developers to self-manage their resources and prototypes using the same platform that is used in production.
- It addresses customer demands in hours or minutes instead of weeks or days, without sacrificing security or performance.
- It reduces operational costs by streamlining operations and automation that optimizes day-to-day tasks and improves employee productivity.

Conclusion

Running Red Hat OpenShift on AWS on Local Zones and Wavelength Zones accelerates edge deployments and offers many benefits. Containers and cloud are the top priorities for CTO/CIO when it comes to digital transformation and innovation for customers. Deploying Red Hat OpenShift on AWS is typically lower cost than traditional on-premises deployments, because you pay only for the infrastructure needed while avoiding the more expensive costs of hosting that infrastructure on-premises.

AWS also offers great flexibility, which allow you to scale hardware resources up or down as required. And, of course, with multiple Regions and Availability Zones, AWS offers a great level of reliability. When coupled with the high availability, orchestration, scalability, and high-performance characteristics, AWS and Red Hat OpenShift together are a winning combination.

Contributors

Contributors to this document include:

- Sankar Panneerselvam, Global GTMS Industry Specialist – Enterprise Digital Platforms, Amazon Web Services – Telco, Media and Technology IBU
- Ovidiu Valeanu – Senior Partner Solutions Architect, Amazon Web Services – RedHat

Further reading

For additional information, see:

- AWS and RedHat joint announcement
- RedHat OpenShift
- RedHat OpenShift on AWS Deployment Guide

Document revisions

Change	Date	Description
Initial publication	September 8, 2022	Whitepaper published

Notices

AWS Glossary

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