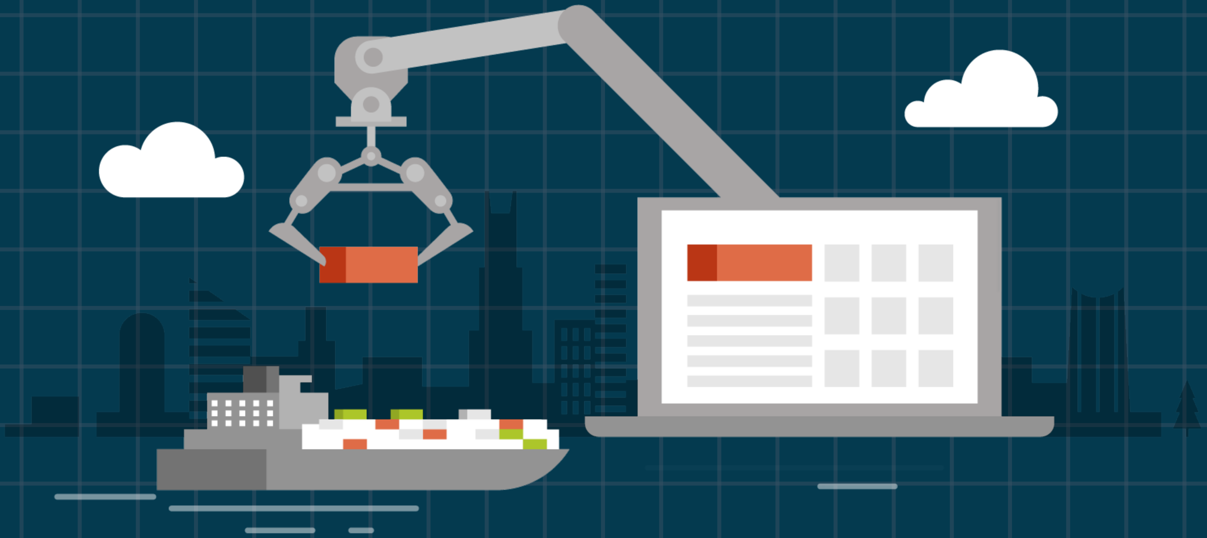
Azure e-book series

Red Hat | Microsoft

Getting started with Azure Red Hat OpenShift

An introduction



Contents

PREFACE

Who this book is for

This guide is meant for developers who are looking to learn how to bolster their application building and deployment capabilities by leveraging Azure and Red Hat OpenShift for full-service deployment of fully managed OpenShift clusters.

What this book covers

In this guide, we will walk you through the ins and outs of using Azure's development tools on the OpenShift platform. We will begin by introducing you to Red Hat OpenShift and the reasons why so many developers and operators choose this as their cluster management platform and how they derive so much utility from it. After learning *why* OpenShift is your preferred platform, you'll learn how to get the most out of it. We'll explain everything you'll need to know about Red Hat OpenShift, as well as how Azure fits into the picture, starting with the fundamental concepts and building blocks of OpenShift. Once you have a solid understanding of the basic OpenShift concepts, a hands-on guide will teach you everything from how to set up your first cluster, to management, to deploying data services, all powered by Azure.

WHAT CAN AZURE DO FOR YOU?

Whether you're a professional developer or just write code for fun, developing with Azure puts the latest cloud technology and best-in-class developer tools at your fingertips, making it easy to build cloud-native applications in your preferred language.

With Azure, you can get work done faster, take your development skills to the next level, and imagine and build tomorrow's applications.

Multiply your impact with:

- A cloud platform
- Developer tools
- Management services

Integrated tightly together, these form a true ecosystem that enable you to create amazing applications and seamless digital experiences that run on any device.

Take advantage of the incredible and always growing capabilities of Azure. Let's dive deep to see what you can do.

INTRODUCTION TO RED HAT OPENSHIFT

Red Hat OpenShift Overview

Red Hat OpenShift is an enterprise-ready Kubernetes container platform with full-stack automated operations to manage hybrid cloud and multi-cloud deployments and is optimized to improve developer productivity and promote innovation. With automated operations and streamlined lifecycle management, RedHat OpenShift empowers development teams to build and deploy new applications and helps operations teams provision, manage, and scale a Kubernetes platforms.

Development teams have access to validated images and solutions from hundreds of partners with security scanning and signing throughout the delivery process. They can access on-demand images and get native access to a wide range of third-party cloud services, all through a single platform.

Operations teams are given visibility into deployments wherever they are, and across teams, with built-in logging and monitoring. Red Hat Kubernetes Operators embed the unique application logic that enables the service to be functional, not simply configured but tuned for performance and updated and patched from the OS with a single touch. Red Hat OpenShift is truly a one-stop-shop that enables organizations to unleash the power of their IT and development teams.

Business Value

OpenShift helps users deliver timely and compelling applications and features across their complex and heterogeneous IT environments and supports key IT initiatives such as containerization, microservices, and cloud migration strategies.

Over 1,000 customers trust Red Hat OpenShift to change the way they deliver applications, improve their relationships with customers, and gain competitive advantages to be leaders in their industries. Per a Forrester Total Economic Impact commissioned study, development teams of the organizations using OpenShift Hosted can better meet business demand and support important IT initiatives, even as they have shifted development cost structures away from IT infrastructure and platform related costs. Benefits seen by these customers include:

- Developers experience a 90% productivity lift for initial application development, testing, and deployment. Developers use OpenShift's templated runtime images, saving days of time and effort for both greenfield projects and legacy application modernization projects. Over three years and a cumulative total of 454 applications, shorter development cycles are worth more than $2.2 million in productivity gains to the composite organization.

- The hosted OpenShift solution reduces elapsed wait time for environment creation by 98%. Red Hat's automated management of environment creation reduces the amount of downtime developers faced before using the solution. Over three years and a cumulative total of 454 applications developed or modernized, developers save 78 hours per application. The organization recaptures 10% of this productivity and establishes a shorter environment creation cycle, worth a cumulative $121,000 over three years.

- Automatic scaling and load balancing managed by Red Hat relieve DevOps and operations teams, providing a 20% lift in operational efficiency. Customers no longer worry about manual scaling by monitoring memory and CPU utilization because the platform autoscales services/pods to fit their computing needs efficiently. Over three years and a cumulative total of 45 employees, the managed services-driven operational benefit is worth $693,000 to the organization.

- Red Hat secures and maintains the platform, saving over 3,000 hours of customer labor per year. Red Hat is responsible for the security, maintenance, and major upgrades of the OpenShift Dedicated platform. Over three years and a cumulative total of 9,270 hours and 24 events, this managed services-driven benefit is worth $332,000 to the organization.

- Operations and administrative costs decrease by 2 FTEs each year. Migrating and modernizing legacy applications using the service improve their availability and performance, reducing the amount of administrative and operational time the organization spent managing legacy applications in place. The shift away from internally managed legacy solutions to OSD deployments is worth more than $974,000 to the organization.

- IT infrastructure cost reductions: Developing on the OpenShift platform requires fewer testing and production servers due to its support of containerization, microservices, and multitenancy, contributing to lower infrastructure costs for interviewed organizations even as their application development efforts expand. Forrester's interviews with three existing customers and subsequent financial analysis found that an organization based on these interviewed organizations experienced benefits of $4.3M over three years versus costs of $981K, adding up to a net present value (NPV) of $3.4M and an ROI of 343%.

Read the full Forrester TEI report for more information on how OpenShift drove significant business results.

Developer productivity lift	**Operations and admin cost savings**	**Automatic scaling and load balancing**	**Security and maintenance efficiencies**	**Reduction in developer wait time**
$2,227,056 over three-year analysis	$974,267 over three-year analysis	$692,654 over three-year analysis	$331,965 over three-year analysis	$120,632 over three-year analysis

What do you get with OpenShift as opposed to Kubernetes?

OpenShift is often referred to as "Enterprise Kubernetes" – but don't let that convince you that they are one and the same. It's also not fair to provide an apples-to-apples comparison of Kubernetes vs. OpenShift, since Kubernetes is an open source project, while OpenShift is an enterprise grade product with a high level of service offerings.

Running containers in production with Kubernetes requires additional tools and resources, such as an image registry, storage management, networking solutions, and logging and monitoring tools, all of which must be versioned and tested together. Building container-based applications requires even more integration work with middleware, frameworks, databases, and CI/CD tools. Azure Red Hat OpenShift combines all this into a single platform, bringing ease of operations to IT teams while giving application teams what they need to execute. All of these topics will be covered in greater detail later in the guide, but with this in mind, let's take a look at some of the key differences between the two.

- **Ease of deployment:** Deploying an application in Kubernetes can be time consuming. This involves pulling your GitHub code onto a machine, spinning up a container, hosting it in a registry like Docker Hub and finally understanding your CI/CD pipeline, which can be very complicated.. OpenShift, on the other hand, automates the heavy lifting and the backend work, only requiring you to create a project and upload your code.

- **Security:** Today, we see that most Kubernetes projects are worked on in teams of multiple developers and operators. Even though Kubernetes now supports things like RBAC and IAM, it still requires a manual setup and configuration, which takes time. Red Hat and OpenShift have done a great job of identifying security best practices after years of experience, which are available to customers out of the box. You simply add new users and OpenShift will handle things like name-spacing and creating different security policies.

- **Flexibility:** In using Azure Red Hat OpenShift, you're able to take advantage of well-known best practices of deployment, management and updating. All the heavy lifting within the backend is taken care of for you without the need for much finger pushing, enabling you to influence your apps quicker. While it's nice for teams that like being told how to get things done and benefit from a streamlined approach, the Kubernetes platform allows you to manually customize your CI/CD DevOps pipeline which offers more room for flexibility and creativity when developing your processes.

- **Day to Day Operations:** Clusters are comprised of a group of multiple VMs and inevitably your operations teams will need to spin up new VMs that need to be added to a cluster. The configuration process through Kubernetes can be time consuming and complex, requiring scripts to be developed to set up things like self-registration or cloud automation. With Azure Red Hat OpenShift, cluster provisioning, scaling, and upgrade operations are automated and managed by the platform.

- **Management:** While you can take advantage of the Kubernetes default dashboards that come with any distribution, most developers need something more robust Azure Red Hat OpenShift offers great web console that builds on the Kubernetes API's and capabilities for operations teams to manage their workloads.

Concepts of OpenShift

Containers

The basic units of Azure Red Hat OpenShift applications are called containers. Linux container technologies are lightweight mechanisms for isolating running processes so that they are limited to interacting with only their designated resources.

Many application instances can be running in containers on a single host without visibility into each other’s processes, files, network, and so on. Typically, each container provides a single service (often called a “micro-service”), such as a web server or a database, though containers can be used for arbitrary workloads.

Images

Containers in Azure Red Hat OpenShift are based on Docker-formatted container images. An image is a binary that includes all the requirements for running a single container, as well as metadata describing its needs and capabilities.

You can think of it as a packaging technology. Containers only have access to resources defined in the image unless you give the container additional access when creating it. By deploying the same image in multiple containers across multiple hosts and load balancing between them, Azure Red Hat OpenShift can provide redundancy and horizontal scaling for a service packaged into an image.

Pods and Services

Azure Red Hat OpenShift leverages the Kubernetes concept of a *pod*, which is one or more container deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed.

Pods are the rough equivalent of a machine instance (physical or virtual) to a container. Each pod is allocated its own internal IP address, therefore owning its entire port space, and containers within pods can share their local storage and networking.

Pods have a life cycle; they are defined, then they are assigned to run on a node, then they run until their container(s) exit or they are removed for some other reason. Pods, depending on policy and exit code, may be removed after exiting, or may be retained in order to enable access to the logs of their containers.

Azure Red Hat OpenShift treats pods as largely immutable; changes cannot be made to a pod definition while it is running. Azure Red Hat OpenShift implements changes by terminating an existing pod and recreating it with modified configuration, base image(s), or both. Pods are also treated as expendable, and do not maintain state when recreated. Therefore, pods should usually be managed by higher-level controllers, rather than directly by users.

Projects and Users

A project is a Kubernetes namespace with additional annotations and is the central vehicle by which access to resources for regular users is managed. A project allows a community of users to organize and manage their content in isolation from other communities. Users must be given access to projects by administrators, or if allowed to create projects, automatically have access to their own projects.

Projects can have a separate **name**, **displayName**, and **description**.

- The mandatory **name** is a unique identifier for the project and is most visible when using the CLI tools or API. The maximum name length is 63 characters.
- The optional **displayName** is how the project is displayed in the web console (defaults to **name**).
- The optional **description** can be a more detailed description of the project and is also visible in the web console.

Developers and administrators can interact with projects using the CLI or the web console.

Builds and Image Streams

A *build* is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image. A BuildConfig object is the definition of the entire build process.

Azure Red Hat OpenShift leverages Kubernetes by creating Docker-formatted containers from build images and pushing them to a container image registry.

Build objects share common characteristics: inputs for a build, the need to complete a build process, logging the build process, publishing resources from successful builds, and publishing the final status of the build. Builds take advantage of resource restrictions, specifying limitations on resources such as CPU usage, memory usage, and build or pod execution time.

The Azure Red Hat OpenShift build system provides extensible support for *build strategies* that are based on selectable types specified in the build API. There are three primary build strategies available:

- Docker build
- Source-to-Image (S2I) build
- Custom build

By default, Docker builds and S2I builds are supported.

The resulting object of a build depends on the builder used to create it. For Docker and S2I builds, the resulting objects are runnable images. For Custom builds, the resulting objects are whatever the builder image author has specified.

Additionally, the Pipeline build strategy can be used to implement sophisticated workflows:

- continuous integration
- continuous deployment

Source-to-Image (S2I)

Source-to-Image (S2I) is a toolkit and workflow for building reproducible container images from source code. S2I produces ready-to-run images by injecting source code into a container image and letting the container prepare that source code for execution. By creating self-assembling builder images, you can version and control your build environments exactly like you use container images to version your runtime environments.

For a dynamic language like Ruby, the build-time and run-time environments are typically the same. Starting with a builder image that describes this environment - with Ruby, Bundler, Rake, Apache, GCC, and other packages needed to set up and run a Ruby application installed - source-to-image performs the following steps:

1. Start a container from the builder image with the application source injected into a known directory
2. The container process transforms that source code into the appropriate runnable setup - in this case, by installing dependencies with Bundler and moving the source code into a directory where Apache has been pre-configured to look for the Ruby config.ru file.
3. Commit the new container and set the image entrypoint to be a script (provided by the builder image) that will start Apache to host the Ruby application.

For compiled languages like C, C++, Go, or Java, the dependencies necessary for compilation might dramatically outweigh the size of the actual runtime artifacts. To keep runtime images slim, S2I enables a multiple-step build processes, where a binary artifact such as an executable or Java WAR file is created in the first builder image, extracted, and injected into a second runtime image that simply places the executable in the correct location for execution.

For example, to create a reproducible build pipeline for Tomcat (the popular Java webserver) and Maven:

1. Create a builder image containing OpenJDK and Tomcat that expects to have a WAR file injected.
2. Create a second image that layers on top of the first image Maven and any other standard dependencies, and expects to have a Maven project injected.
3. Invoke source-to-image using the Java application source and the Maven image to create the desired application WAR.
4. Invoke source-to-image a second time using the WAR file from the previous step and the initial Tomcat image to create the runtime image.

By placing our build logic inside of images, and by combining the images into multiple steps, we can keep our runtime environment close to our build environment (same JDK, same Tomcat JARs) without requiring build tools to be deployed to production.

The goals and benefits of using Source-To-Image (S2I) as your build strategy are:

- **Reproducibility:** Allow build environments to be tightly versioned by encapsulating them within a container image and defining a simple interface (injected source code) for callers. Reproducible builds are a key requirement to enabling security updates and continuous integration in containerized infra-structure, and builder images help ensure repeatability as well as the ability to swap runtimes.

- **Flexibility:** Any existing build system that can run on Linux can be run inside of a container, and each individual builder can also be part of a larger pipeline. In addition, the scripts that process the application source code can be injected into the builder image, allowing authors to adapt existing images to enable source handling.

- **Speed:** Instead of building multiple layers in a single Docker file, S2I encourages authors to represent an application in a single image layer. This saves time during creation and deployment and allows for better control over the output of the final image.

- **Security:** Docker files are run without many of the normal operational controls of containers, usually running as root and having access to the container network. S2I can be used to control what permissions and privileges are available to the builder image since the build is launched in a single container. In concert with platforms like OpenShift, source-to-image can enable admins to tightly control what privileges developers have at build time.

Replication Controllers

A replication controller ensures that a specified number of replicas of a pod are running at all times. If pods exit or are deleted, the replication controller acts to instantiate more, up to the defined number. Likewise, if there are more running than desired, it deletes as many as necessary to match the defined amount.

A replication controller configuration consists of:

- The number of replicas desired (which can be adjusted at runtime).
- A pod definition to use when creating a replicated pod.
- A selector for identifying managed pods.
- A selector is a set of labels assigned to the pods that are managed by the replication controller. These labels are included in the pod definition that the replication controller instantiates. The replication controller uses the selector to determine how many instances of the pod are already running in order to adjust as needed.

The replication controller does not perform auto-scaling based on load or traffic, as it does not track either. Rather, this would require its replica count to be adjusted by an external auto-scaler.

A replication controller is a core Kubernetes object called **`ReplicationController`**. The following is an example **`ReplicationController`** definition:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1  (1)
  selector:    (2)
    name: frontend
  template:    (3)
    metadata:
      labels:  (4)
        name: frontend (5)
    spec:
      containers:
        - image: openshift/hello-openshift
          name: helloworld
          ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
```

__(1)__ The number of copies of the pod to run. __(2)__ The label selector of the pod to run. __(3)__ A template for the pod the controller creates. __(4)__ Labels on the pod should include those from the label selector. __(5)__ The maximum name length after expanding any parameters is 63 characters.

Replica Set

Similar to a replication controller, a replica set ensures that a specified number of pod replicas are running at any given time. The difference between a replica set and a replication controller is that a replica set supports set-based selector requirements whereas a replication controller only supports equality-based selector requirements.

Only use replica sets if you require custom update orchestration or do not require updates at all, otherwise, use Deployments. Replica sets can be used independently, but are used by deployments to orchestrate pod creation, deletion, and updates. Deployments manage their replica sets automatically, provide declarative updates to pods, and do not have to manually manage the replica sets that they create.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
   name: frontend-1
   labels:
      tier: frontend
spec:
   replicas: 3
   selector: (1)
      matchLabels: (2)
         tier: frontend
      matchExpressions: (3)
      - {key: tier, operator: In, values: [frontend]}
   template:
      metadata:
         labels:
           tier: frontend
      spec:
         containers:
         - image: openshift/hello-openshift
           name : helloworld
           ports:
           - containerPort: 8080
              protocol: TCP
         restartPolicy: Always
```

(1) *A label query over a set of resources. The result of* `matchLabels` *and* `matchExpressions` *are logically conjoined.* ***(2)*** *Equality-based selector to specify resources with labels that match the selector.* ***(3)*** *Set-based selector to filter keys. This selects all resources with key equal to* `tier` *and value equal to* `frontend`*.*

Jobs

A job is similar to a replication controller, in that its purpose is to create pods for specific reasons. The difference is that replication controllers are designed for pods that will be continuously running, whereas jobs are for one-time pods. A job tracks any successful completions and when the specified amount of completions have been reached, the job itself is completed.

The following example computes π to 2000 places, prints it out, then completes:

```
apiVersion: extensions/v1
kind: Job
metadata:
   name: pi
spec:
   selector:
      matchLabels:
        app: pi
   template:
      metadata:
        name: pi
        labels:
        app: pi
   spec:
      containers:
      - name: pi
        image: perl
        command:  [“perl”, “-Mbignum=bpi”, “-wle”,
                  “print bpi(2000)”]
   restartPolicy: Never
```

See the Jobs topic for more information on how to use jobs.

Deployments and Deployment Configurations

Building on replication controllers, Azure Red Hat OpenShift adds expanded support for the software development and deployment lifecycle with the concept of deployments. In the simplest case, a deployment just creates a new replication controller and lets it start up pods. However, Azure Red Hat OpenShift deployments also provide the ability to transition from an existing deployment of an image to a new one and also define hooks to be run before or after creating the replication controller.

The Azure Red Hat OpenShift `DeploymentConfig` object defines the following details of a deployment:

1. The elements of a `ReplicationController` definition.
2. Triggers for creating a new deployment automatically.
3. The strategy for transitioning between deployments.
4. Life cycle hooks.

Each time a deployment is triggered, whether manually or automatically, a deployer pod manages the deployment (including scaling down the old replication controller, scaling up the new one, and running hooks). The deployment pod remains for an indefinite amount of time after it completes the deployment in order to retain its logs of the deployment. When a deployment is superseded by another, the previous replication controller is retained to enable easy rollback if needed.

For detailed instructions on how to create and interact with deployments, refer to Deployments.

Here is an example **DeploymentConfig** definition with some omissions and callouts:

```
apiVersion: v1
kind: DeploymentConfig
metadata:
   name: frontend
spec:
   replicas: 5
   selector:
      name: frontend
   template: { ... }
   triggers:
   - type: ConfigChange (1)
   - imageChangeParams:
        automatic: true
        containerNames:
        - helloworld
        from:
           kind: ImageStreamTag
           name: hello-openshift:latest
      type: ImageChange (2)
   strategy:
      type: Rolling (3)
```

(1) *A* **ConfigChange** *trigger causes a new deployment to be created any time the replication controller template changes.* ***(2)*** *An* **ImageChange** *trigger causes a new deployment to be created each time a new version of the backing image is available in the named image stream.* ***(3)*** *The default* **Rolling** *strategy makes a downtime-free transition between deployments.*

Routes

An Azure Red Hat OpenShift route exposes a service at a host name, such as *www.example.com*, so that external clients can reach it by name.

DNS resolution for a host name is handled separately from routing. Your administrator may have configured a DNS wildcard entry that will resolve to the Azure Red Hat OpenShift node that is running the Azure Red Hat OpenShift router. If you are using a different host name you may need to modify its DNS records independently to resolve to the node that is running the router.

Each route consists of a name (limited to 63 characters), a service selector, and an optional security configuration.

Templates

A template describes a set of objects that can be parameterized and processed to produce a list of objects for creation by Azure Red Hat OpenShift. A template can be processed to create anything you have permission to create within a project, for example services, build configurations, and deployment configurations. A template may also define a set of labels to apply to every object defined in the template.

You can create a list of objects from a template using the CLI or, if a template has been uploaded to your project or the global template library, using the web console. For a curated set of templates, see the OpenShift Image Streams and Templates library.

AZURE RED HAT OPENSHIFT

With Azure Red Hat OpenShift, you can deploy fully managed Red Hat OpenShift clusters without worrying about building and managing the infrastructure to run it.

Running containers in production with Kubernetes requires additional tools and resources, such as an image registry, storage management, networking solutions, and logging and monitoring tools, all of which must be versioned and tested together. Building container-based applications requires even more integration work with middleware, frameworks, databases, and CI/CD tools. Azure Red Hat OpenShift extends Kubernetes and combines all this into a single platform, bringing ease of operations to IT teams while giving application teams what they need to execute.

Azure Red Hat OpenShift is jointly engineered, operated, and supported by Red Hat and Microsoft to provide an integrated support experience. There are no virtual machines to operate, and no patching is required. Master, infrastructure and application nodes are patched, updated, and monitored on your behalf by Red Hat and Microsoft. Your Azure Red Hat OpenShift clusters are deployed into your Azure subscription and are included on your Azure bill.

You can choose your own registry, networking, storage, and CI/CD solutions, or use the built-in solutions for automated source code management, container and application builds, deployments, scaling, health management, and more. Azure Red Hat OpenShift provides an integrated sign-on experience through Azure Active Directory.

In just minutes, deploy Red Hat OpenShift clusters on Azure for:

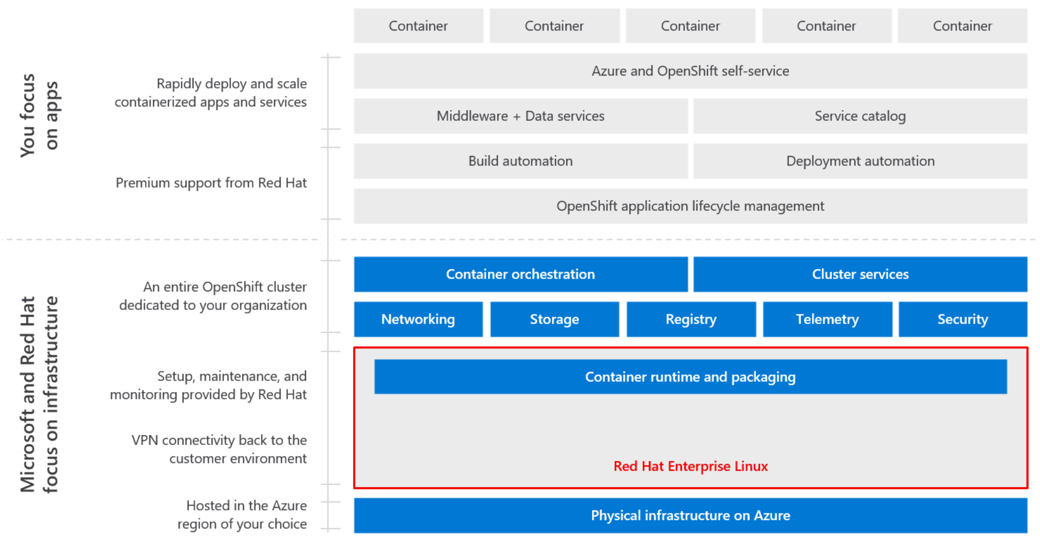
- Enterprise grade operations, security and compliance with an integrated support experience.
- Empowering developers to innovate with productivity through built-in CI/CD pipelines, then easily connect your applications to hundreds of Azure services such as MySQL, PostgreSQL, Redis, Azure Cosmos DB, and more.
- Scalability on your terms where you can start a highly available cluster with four application nodes in a few minutes, then scale as your application demand changes; plus, get your choice of standard, high-memory, or high-CPU application nodes.

Architecture

Azure Red Hat OpenShift has a microservices-based architecture of smaller, decoupled units that work together. It runs on top of a Kubernetes cluster, with data about the objects stored in etcd, a reliable clustered key-value store. Those services are broken down by function:

- REST APIs, which expose each of the core objects.
- Controllers, which read those APIs, apply changes to other objects, and report status or write back to the object.

Users make calls to the REST API to change the state of the system. Controllers use the REST API to read the user's desired state, and then try to bring the other parts of the system into sync. For example, when a user requests a build they create a "build" object. The build controller sees that a new build has been created, and runs a process on the cluster to perform that build. When the build completes, the controller updates the build object via the REST API and the user sees that their build is complete.


You focus on apps
Rapidly deploy and scale containerized apps and services
Premium support from Red Hat
Microsoft and Red Hat focus on infrastructure
An entire OpenShift cluster dedicated to your organization
Setup, maintenance, and monitoring provided by Red Hat
VPN connectivity back to the customer environment
Hosted in the Azure region of your choice
Container
Container
Container
Container
Container
Azure and OpenShift self-service
Middleware + Data services
Service catalog
Build automation
Deployment automation
OpenShift application lifecycle management
Container orchestration
Cluster services
Networking
Storage
Registry
Telemetry
Security
Container runtime and packaging
Red Hat Enterprise Linux
Physical infrastructure on Azure

To make this possible, controllers leverage a reliable stream of changes to the system to sync their view of the system with what users are doing. This event stream pushes changes from etcd to the REST API and then to the controllers as soon as changes occur, so changes can ripple out through the system very quickly and efficiently. However, since failures can occur at any time, the controllers must also be able to get the latest state of the system at startup, and confirm that everything is in the right state. This resynchronization is important, because it means that even if something goes wrong, the operator can restart the affected components and the system double checks everything before continuing. The system should eventually converge to the user's intent, since the controllers can always bring the system into sync.

Within Azure Red Hat OpenShift, Kubernetes manages containerized applications across a set of containers or hosts and provides mechanisms for deployment, maintenance, and application-scaling. The container runtime packages, instantiates, and runs containerized applications. A Kubernetes cluster consists of one or more masters and a set of nodes.

Master, infrastructure and application nodes

The master nodes are hosts that contain the control plane components, including the API server, controller manager server, and etcd. The masters manage nodes in its Kubernetes cluster and schedules pods to run on those nodes.

A node provides the runtime environments for containers. Each node in a Kubernetes cluster has the required services to be managed by the master. Nodes also have the required services to run pods, including the container runtime, a kubelet, and a service proxy.

Each node also runs a simple network proxy that reflects the services defined in the API on that node. This allows the node to do simple TCP and UDP stream forwarding across a set of back ends.

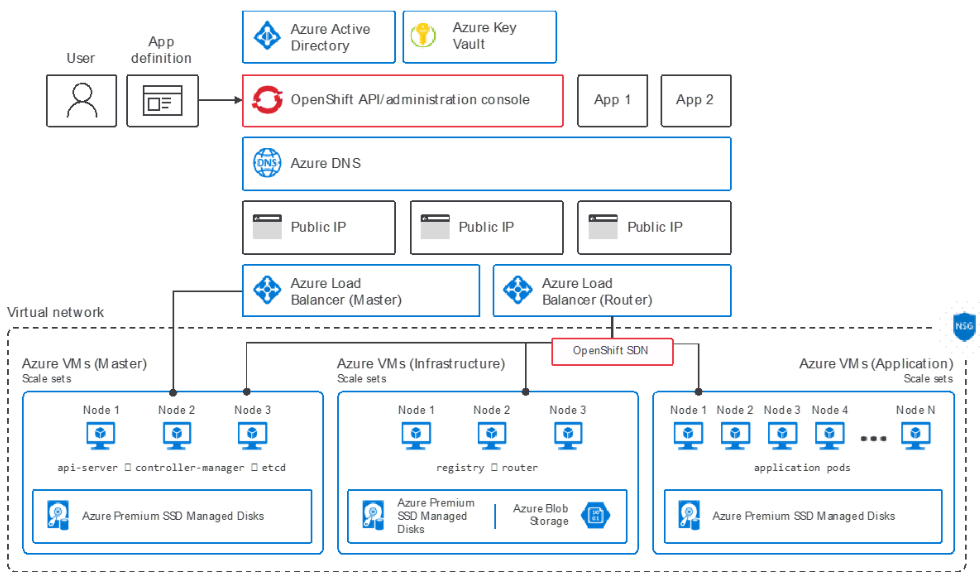
Azure Red Hat OpenShift creates nodes that run on Azure Virtual Machines that are connected to Azure Premium SSD disks for storage.

Container registry

Azure Red Hat OpenShift provides an integrated container image registry called OpenShift Container Registry (OCR) that adds the ability to automatically provision new image repositories on demand. This provides users with a built-in location for their application builds to push the resulting images.

Whenever a new image is pushed to OCR, the registry notifies Azure Red Hat OpenShift about the new image, passing along all the information about it, such as the namespace, name, and image metadata. Different pieces of Azure Red Hat OpenShift react to new images, creating new builds and deployments.

Azure Red Hat OpenShift can also utilize any server implementing the container image registry API as a source of images, including the Docker Hub and Azure Container Registry.

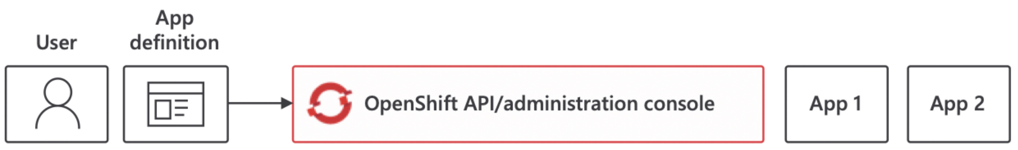


Management

As a managed service, Microsoft and Red Hat:

- Manage and monitor all the underlying virtual machines and infrastructure
- Manage environment patches
- Secure the cluster

So that you can focus on what matters most, developing great applications.



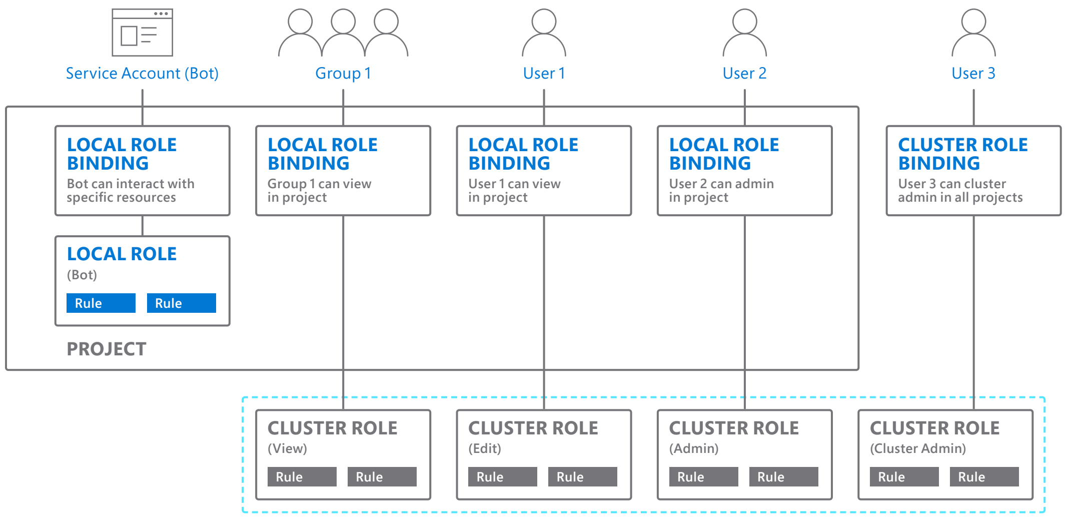
Security

The Azure Red Hat OpenShift and Kubernetes APIs authenticate users who present credentials via Azure Active Directory (Azure AD) integration, and then authorize them based on their role.

The authentication layer identifies the user associated with requests to the Azure Red Hat OpenShift API. The authorization layer then uses information about the requesting user to determine if the request should be allowed.

Authorization is handled in the Azure Red Hat OpenShift policy engine, which defines actions like "create pod" or "list services" and groups them into roles in a policy document. Roles are bound to users or groups by the user or group identifier. When a user or service account attempts an action, the policy engine checks for one or more of the roles assigned to the user (e.g., customer administrator or administrator of the current project) before allowing it to continue.

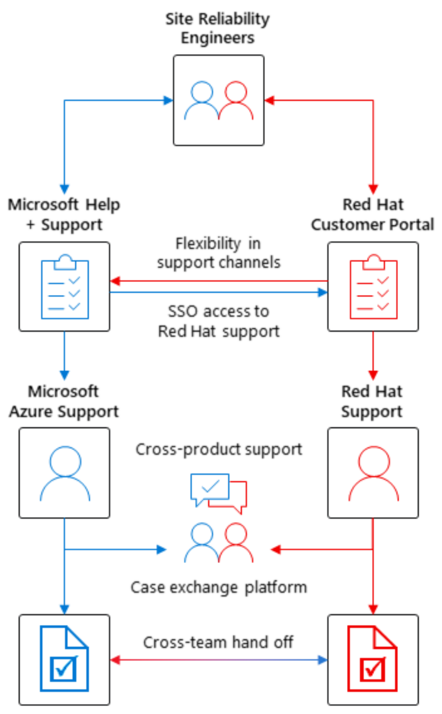
The relationships between cluster roles, local roles, cluster role bindings, local role bindings, users, groups and service accounts are illustrated below.



Support

Azure Red Hat OpenShift is unique in the way support is managed. Microsoft and Red Hat Site Reliability Engineers (SREs) work together ensuring the smooth operation of the service.

Customers request support in the Azure portal, and the requests are triaged and addressed by Microsoft and Red Hat engineers to quickly address customer support requests, whether those are at the Azure platform level or at the OpenShift level.



SETTING UP THE CLUSTER, NETWORKING AND SECURITY

Install the Azure CLI and sign in to Azure

Install the Azure CLI

You'll need to run Azure CLI commands to provision the cluster. The Azure CLI is a command-line tool providing a great experience for managing Azure resources. The CLI is designed to make scripting easy, query data, support long-running operations, and more.

Azure Red Hat OpenShift requires version 2.0.65 or higher of the Azure CLI. If you've already installed the Azure CLI, you can check which version you have by running:

```
az --version
```

The first line of output will have the CLI version, for example azure-cli (2.0.65).

Alternatively, you can use the Azure Cloud Shell. When using the Azure Cloud Shell, be sure to select the Bash environment.

Sign in to Azure

If you're running the Azure CLI locally, open a Bash command shell and run az login to sign in to Azure.

```
az login
```

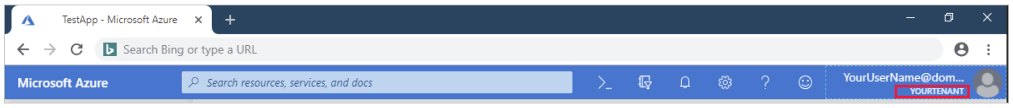
If you have access to multiple subscriptions, run az account set -s {subscription ID} replacing {subscription ID} with the subscription you want to use.

Create an Azure Active Directory tenant for your cluster

Microsoft Azure Red Hat OpenShift requires an Azure Active Directory (Azure AD) tenant in which to create your cluster. A *tenant* is a dedicated instance of Azure AD that an organization or app developer receives when they create a relationship with Microsoft by signing up for Azure, Microsoft Intune, or Microsoft 365. Each Azure AD tenant is distinct and separate from other Azure AD tenants and has its own work and school identities and app registrations.

If you don't already have an Azure AD tenant, follow these instructions to create one, otherwise, you can skip to creating the administrator user and administrator security group.

1. Sign in to the Azure portal using the account you wish to associate with your Azure Red Hat OpenShift cluster.
2. Open the Azure Active Directory blade to create a new tenant (also known as a new *Azure Active Directory*).
3. Provide an Organization name.
4. Provide an Initial domain name. This will have onmicrosoft.com appended to it. You can reuse the value for *Organization name* here.
5. Choose a country or region where the tenant will be created.
6. Click Create.
7. After your Azure AD tenant is created, select the Click here to manage your new directory link. Your new tenant name should be displayed in the upper-right of the Azure portal:


TestApp - Microsoft Azure
Search Bing or type a URL
Microsoft Azure
Search resources, services, and docs
YourUserName@dom...
YOURTENANT

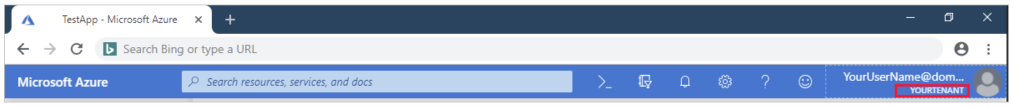
8. Make note of the tenant ID so you can later specify where to create your Azure Red Hat OpenShift cluster. In the portal, you should now see the Azure Active Directory overview blade for your new tenant. Select Properties and copy the value for your Directory ID. We will refer to this value as **{tenant id}** in the Create an Azure Red Hat OpenShift cluster section.

Create the administrator user and administrator security group

Microsoft Azure Red Hat OpenShift needs permissions to perform tasks on behalf of your cluster. If your organization doesn't already have an Azure Active Directory (Azure AD) user, Azure AD security group, or an Azure AD app registration to use as the service principal, follow these instructions to create them.

Create a new Azure Active Directory user

In the Azure portal, ensure that your tenant appears under your user name in the top right of the portal:


TestApp - Microsoft Azure
Search Bing or type a URL
Microsoft Azure
Search resources, services, and docs
YourUserName@dom...
YOURTENANT

If the wrong tenant is displayed, click your user name in the top right, then click Switch Directory, and select the correct tenant from the All Directories list. Create a new Azure Active Directory global administrator user to sign in to your Azure Red Hat OpenShift cluster.

1. Go to the Users-All users blade.
2. Click +New user to open the User pane.
3. Enter a Name for this user.
4. Create a User name based on the name of the tenant you created, with .onmicrosoft.com appended at the end.
   For example, yourUserName@yourTenantName.onmicrosoft.com.
   Write down this user name. You'll need it to sign into your cluster.
5. Click Directory role to open the directory role pane, and select Global administrator and then click Ok at the bottom of the pane.
6. In the User pane, click Show Password and record the temporary password. After you sign in the first time, you'll be prompted to reset it.
7. At the bottom of the pane, click Create to create the user.

Create a new Azure Active Directory security group

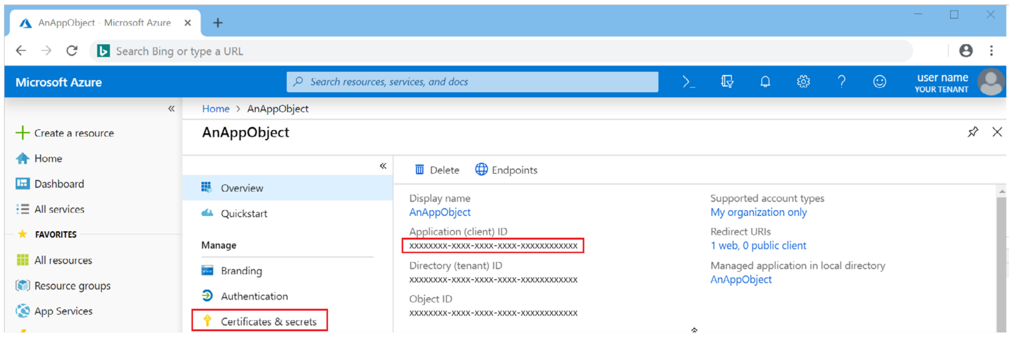
To grant cluster admin access, memberships in an Azure AD security group are synced into the OpenShift group "osa-customer-admins". If not specified, no cluster admin access will be granted.

1. Open the Azure Active Directory groups blade.
2. Click +New Group.
3. Provide a group name and description.
4. Set Group type to Security.
5. Set Membership type to Assigned. Add the Azure AD user that you created in the earlier step to this security group.
6. Click Members to open the Select members pane.
7. In the members list, select the Azure AD user that you created above.
8. At the bottom of the portal, click on Select and then Create to create the security group. Write down the Group ID value.
9. When the group is created, you will see it in the list of all groups. Click on the new group.
10. On the page that appears, copy down the Object ID. We will refer to this value as **{group id}** in the Create an Azure Red Hat OpenShift cluster section.

Create an Azure Active Directory app registration for authentication

If your organization doesn't already have an Azure Active Directory (Azure AD) app registration to use as a service principal, follow these instructions to create one.

1. Open the [App registrations blade]() and click +New registration.
2. In the Register an application pane, enter a name for your application registration.
3. Ensure that under Supported account types that Accounts in this organizational directory only is selected. This is the most secure choice.
4. We will add a redirect URI later once we know the URI of the cluster. Click the Register button to create the Azure AD application registration.
5. On the page that appears, copy down the Application (client) ID. We will refer to this value as **{app id}** in the [Create an Azure Red Hat OpenShift cluster]() section.



Display name
AnAppObject

Application (client) ID
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Directory (tenant) ID
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Object ID
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Supported account types
My organization only

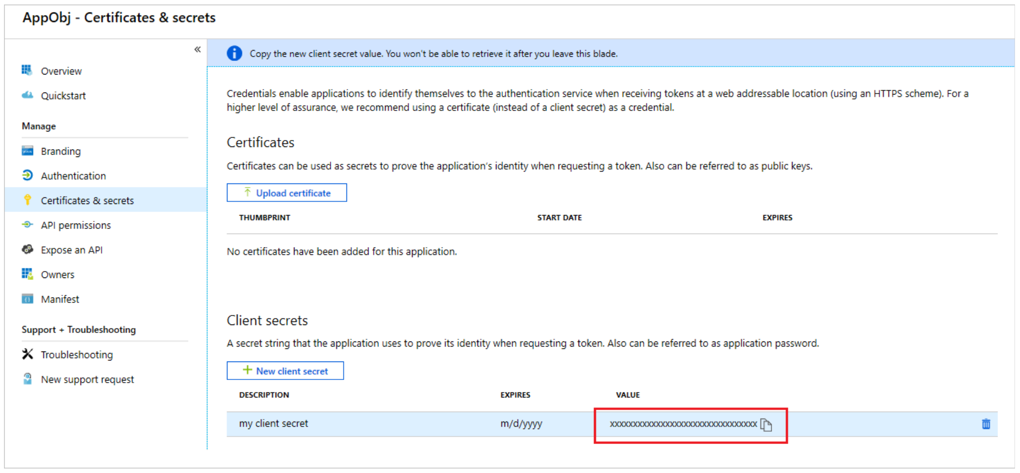
Redirect URIs
1 web, 0 public client

Managed application in local directory
AnAppObject

Create a client secret

Generate a client secret for authenticating your app to Azure Active Directory.

1. In the Manage section of the app registrations page, click Certificates & secrets.
2. On the Certificates & secrets pane, click +New client secret. The Add a client secret pane appears.
3. Provide a Description.
4. Set Expires to the duration you prefer, for example In 2 Years.
5. Click Add and the key value will appear in the Client secrets section of the page.
6. Copy down the key value. We will refer to this value as **{secret}** in the Create an Azure Red Hat OpenShift cluster section.



Copy the new client secret value. You won't be able to retrieve it after you leave this blade.

Credentials enable applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

THUMBPRINT	START DATE	EXPIRES

No certificates have been added for this application.

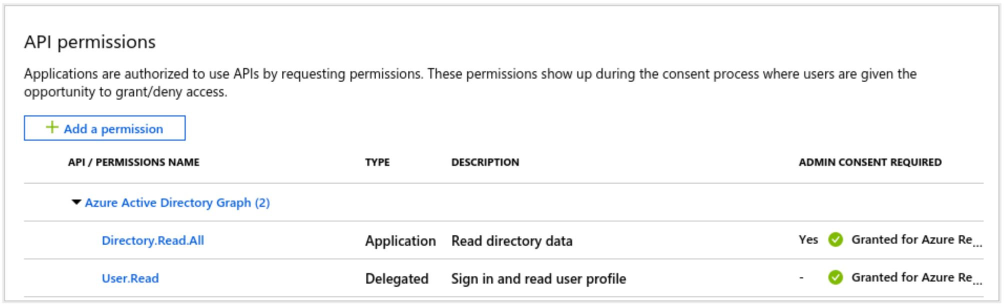
Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

DESCRIPTION	EXPIRES	VALUE
my client secret	m/d/yyyy	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Add API permissions

1. In the Manage section click API permissions.
2. Click Add permission and select Azure Active Directory Graph then Delegated permissions.
3. Expand User on the list below and make sure User.Read is enabled.
4. Scroll up and select Application permissions.
5. Expand Directory on the list below and enable Directory.ReadAll
6. Click Add permissions to accept the changes.
7. The API permissions panel should now show both *User.Read* and *Directory.ReadAll*. Please note the warning in Admin consent required column next to *Directory.ReadAll*.
8. If you are the *Azure Subscription Administrator*, click Grant admin consent for Subscription Name below. If you are not the *Azure Subscription Administrator*, request the consent from your administrator.



API permissions

Applications are authorized to use APIs by requesting permissions. These permissions show up during the consent process where users are given the opportunity to grant/deny access.

API / PERMISSIONS NAME	TYPE	DESCRIPTION	ADMIN CONSENT REQUIRED	
Azure Active Directory Graph (2)				
Directory.Read.All	Application	Read directory data	Yes	✓ Granted for Azure Re...
User.Read	Delegated	Sign in and read user profile	-	✓ Granted for Azure Re...

Restrict the cluster access to assigned users and assign user access

Applications registered in an Azure Active Directory (Azure AD) tenant are, by default, available to all users of the tenant who authenticate successfully. Azure AD allows tenant administrators and developers to restrict an app to a specific set of users or security groups in the tenant.

Update the app to enable user assignment

1. Go to the Azure portal and sign-in as a Global Administrator.
2. On the top bar, select the signed-in account.
3. Under Directory, select the Azure AD tenant where the app will be registered.
4. In the navigation on the left, select Azure Active Directory. If Azure Active Directory is not available in the navigation pane, then follow these steps:
    a. Select All services at the top of the main left-hand navigation menu.
    b. Type in Azure Active Directory in the filter search box and then select the Azure Active Directory item from the result.
5. In the Azure Active Directory pane, select Enterprise Applications from the Azure Active Directory left-hand navigation menu.
6. Select All Applications to view a list of all your applications. If you do not see the application you want, use the various filters at the top of the All applications list to restrict the list or scroll down the list to locate your application.
7. Select the application you want to assign a user or security group to from the list.
8. In the application's Overview page, select Properties from the application's left-hand navigation menu.
9. Locate the setting User assignment required? and set it to Yes. When this option is set to Yes, then users must first be assigned to this application before being able to access it.
10. Select Save to save this configuration change.

Assign users and groups to the app

Once you've configured your app to enable user assignment, you can go ahead and assign users and groups to the app.

1. Select the Users and groups pane in the application's left-hand navigation menu.
2. At the top of the Users and groups list, select the Add user button to open the Add Assignment pane.
3. Select the Users selector from the Add Assignment pane. A list of users and security groups will be shown along with a textbox to search and locate a certain user or group. This screen allows you to select multiple users and groups in one go.
4. Once you are done selecting the users and groups, press the Select button on bottom to move to the next part.
5. Press the Assign button on the bottom to finish the assignments of users and groups to the app.
6. Confirm that the users and groups you added are showing up in the updated Users and groups list.

Create the cluster and connect it to your existing Virtual Network

Register providers and features

The Microsoft.ContainerService AROGA feature, Microsoft.Solutions , Microsoft.Compute , Microsoft.Storage , Microsoft.KeyVault, and Microsoft.Network providers must be registered to your subscription manually before deploying your first Azure Red Hat OpenShift cluster.

To register these providers and features manually, use the following instructions from a Bash shell if you've installed the CLI, or from the Azure Cloud Shell (Bash) session in your Azure portal:

1. If you have multiple Azure subscriptions, specify the relevant subscription ID:
   ```
   az account set --subscription <SUBSCRIPTION ID>
   ```
2. Register the Microsoft.ContainerService AROGA feature:
   ```
   az feature register --namespace Microsoft.ContainerService -n
   AROGA
   ```
3. Register the Microsoft.Storage provider:
   ```
   az provider register -n Microsoft.Storage --wait
   ```
4. Register the Microsoft.Compute provider:
   ```
   az provider register -n Microsoft.Compute --wait
   ```
5. Register the Microsoft.Solutions provider:
   ```
   az provider register -n Microsoft.Solutions --wait
   ```
6. Register the Microsoft.Network provider:
   ```
   az provider register -n Microsoft.Network --wait
   ```
7. Register the Microsoft.KeyVault provider:
   ```
   az provider register -n Microsoft.KeyVault --wait
   ```
8. Refresh the registration of the Microsoft.ContainerService resource provider:
   ```
   az provider register -n Microsoft.ContainerService --wait
   ```

Retrieve the peering Virtual Network details

If you don’t need to connect the virtual network (VNET) of the cluster you create to an existing VNET via peering, skip this step.

If peering to a network outside the default subscription then in that subscription, you will also need to register the provider Microsoft.ContainerService. To do this, run the below command in that subscription. Else, if the VNET you are peering is located in the same subscription, you can skip the registering step.

```
az provider register -n Microsoft.ContainerService --wait
```

First, get the identifier of the existing VNET. The identifier will be of the form: /subscriptions/{subscription id}/resourceGroups/{resource group of VNET}/providers/Microsoft.Network/virtualNetworks/{VNET name}.

If you don’t know the network name or the resource group the existing VNET belongs to, go to the Virtual networks blade and click on your virtual network. The Virtual network page appears and will list the name of the network and the resource group it belongs to.

Retrieve the Virtual Network ID variable using the following CLI command in a Bash shell. You’ll refer to its value as **{peering vnet id}**.

```
az network vnet show -n {VNET name} -g {VNET resource group} --query
id -o tsv
```

Create the resource group

Create the resource group for the cluster. Specify the resource group name and location.

```
az group create --name {resource group name} --location {location}
```

Create the cluster

You're now ready to create a cluster. The following will create the cluster in the specified Azure AD tenant, specify the Azure AD app object and secret to use as a security principal, and the security group that contains the members that have admin access to the cluster.

Make sure to replace the **{app id}**, **{secret}**, **{tenant id}**, **{group id}** and {peering vnet id} with the values you made note of before.

```
az openshift create --resource-group {resource group name} --name
{cluster name} --location {location} --aad-client-app-id {app id}
--aad-client-app-secret {secret} --aad-tenant-id {tenant id}
--customer-admin-group-id {group id} --vnet-peer {peering vnet id}
```

After a few minutes, az openshift create will complete.

Update your app registration redirect URI

To be able to login to the cluster, you'll need to update the app registration you created in the Create an Azure Active Directory app registration for authentication step with the sign in URL of your cluster. This will enable Azure Active Directory authentication to properly redirect back to your cluster's web console after successful authentication.

Get the sign in URL for your cluster

```
az openshift show -n {cluster name} -g {resource group name} --query
"publicHostname" -o tsv
```

You should get back something like openshift.xxxxxxxxxxxxxxxxxxxx.eastus.azmosa.io. The sign in URL for your cluster will be https:// followed by the publicHostName value.

For example: https://openshift.xxxxxxxxxxxxxxxxxxxx.eastus.azmosa.io.
You will use this URI in the next step as part of the app registration redirect URI.
Now that you have the sign in URL for the cluster, set the app registration redirect UI:

1. Open the App registrations blade.
2. Click on your app registration object.
3. Click on Add a redirect URI.
4. Ensure that TYPE is Web and set the REDIRECT URI using the following pattern: https://<public host name>/oauth2callback/Azure%20AD.
   For example:
   https://openshift.xxxxxxxxxxxxxxxxxxxx.eastus.azmosa.io/oauth2callback/Azure%20AD
5. Click Save.

ACCESSING THE CLUSTER

Via the Web UI

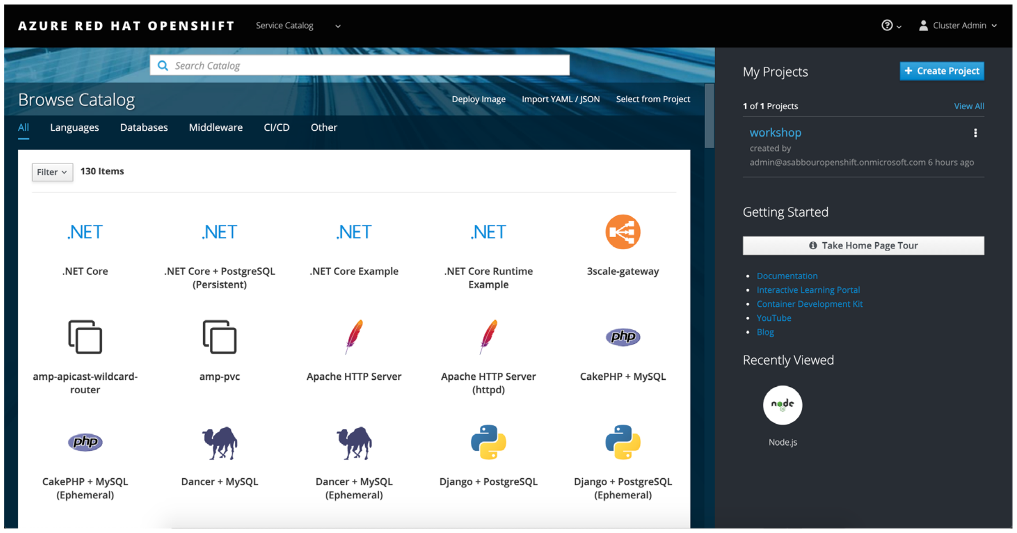
From a Bash shell if you've installed the CLI, or from the Azure Cloud Shell (Bash) session in your Azure portal, retrieve your cluster sign in URL by running:

```
az openshift show -n {cluster name} -g {resource group name} --query
“publicHostname” -o tsv
```

You should get back something like openshift.xxxxxxxxxxxxxxxxxxxx.eastus.azmosa.io. The sign in URL for your cluster will be https:// followed by the publicHostName value. For example: https://openshift.xxxxxxxxxxxxxxxxxxxx.eastus.azmosa.io.

Open this URL in your browser, you'll be asked to login with Azure Active Directory. Use the username and password for the user you created.

After logging in, you should be able to see the Azure Red Hat OpenShift Web Console.



Via OpenShift CLI (oc)

Downloading the OpenShift CLI

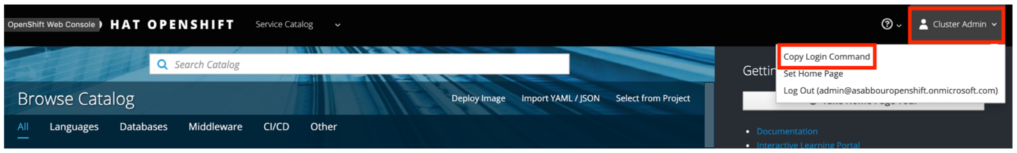
You'll need to download the latest OpenShift CLI (oc) client tools release for OpenShift 3.11.

From a Bash shell if you've installed the CLI, or from the Azure Cloud Shell (Bash) session in your Azure portal, download the latest release, extract it into the openshift directory, then make it available on your PATH.

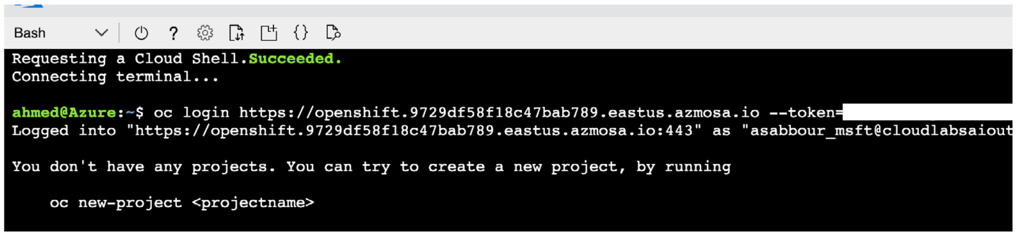
```
wget
https://github.com/openshift/origin/releases/download/v3.11.0/openshif
t-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz

mkdir openshift

tar -zxvf
openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz -C
openshift --strip-components=1

echo 'export PATH=$PATH:~/openshift' >> ~/.bashrc && source ~/.bashrc
```

Running the OpenShift CLI and logging into your cluster

To authenticate against your cluster, you'll need to retrieve the login command and token from the Web Console. Once you're logged into the Web Console, click on the username on the top right, then click Copy login command, which will look something like oc login https://openshift.xxxxxxxxxxxxxxxxxxxx.eastus.azmosa.io --token=[authentication token]



From a Bash shell if you've installed the CLI, or from the Azure Cloud Shell (Bash) session in your Azure portal, paste that login command and you should be able to connect to your cluster.

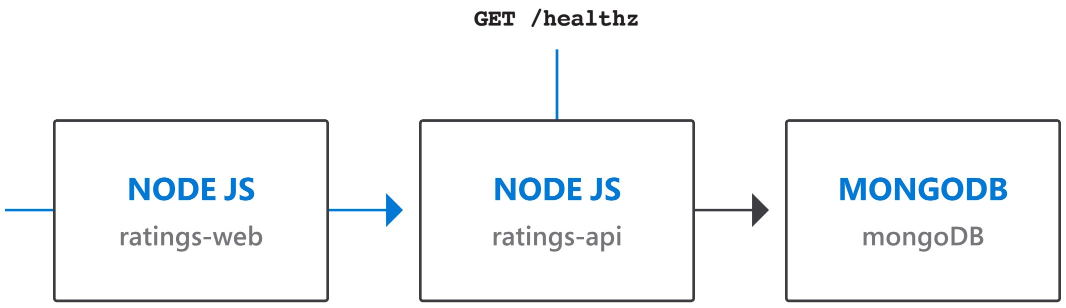


```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

ahmed@Azure:~$ oc login https://openshift.9729df58f18c47bab789.eastus.azmosa.io --token=[redacted]
Logged into "https://openshift.9729df58f18c47bab789.eastus.azmosa.io:443" as "asabbour_msft@cloudlabsaiout

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>
```

CREATING A MULTI-CONTAINER RATINGS APPLICATION

Application Overview

In this chapter, you will be deploying a ratings application on Azure Red Hat OpenShift. The application consists of a frontend container and an API container, both written in NodeJS. The API reads/writes data to a MongoDB.



You can find the application source code in the links below.

Component	Link
A public facing API rating-api	GitHub repo
A public facing web frontend rating-web	GitHub repo

Connect to the cluster and create a project

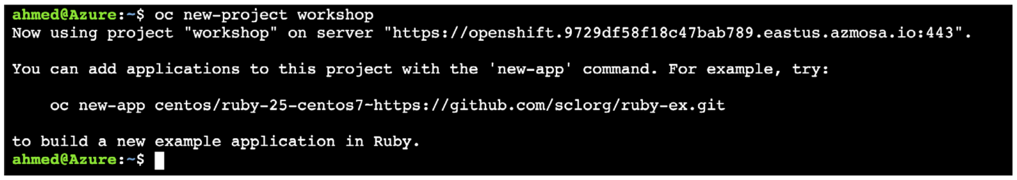
Connect and authenticate against the cluster

Follow the steps in accessing the cluster section to download the OpenShift CLI and authenticate against the cluster.

Create the project

A project allows a community of users to organize and manage their content in isolation from other communities.

```
oc new-project workshop
```



```
ahmed@Azure:~$ oc new-project workshop
Now using project "workshop" on server "https://openshift.9729df58f18c47bab789.eastus.azmosa.io:443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
ahmed@Azure:~$ 
```

Deploy MongoDB

Azure Red Hat OpenShift provides a container image and template to make creating a new MongoDB database service easy. The template provides parameter fields to define all the mandatory environment variables (user, password, database name, etc) with predefined defaults including auto-generation of password values. It will also define both a deployment configuration and a service.

Create MongoDB from template

There are two templates available:

- mongodb-ephemeral is for development/testing purposes only because it uses ephemeral storage for the database content. This means that if the database pod is restarted for any reason, such as the pod being moved to another node or the deployment configuration being updated and triggering a redeploy, all data will be lost.
- mongodb-persistent uses a persistent volume store for the database data which means the data will survive a pod restart. Using persistent volumes requires a persistent volume pool be defined in the Azure Red Hat OpenShift deployment.

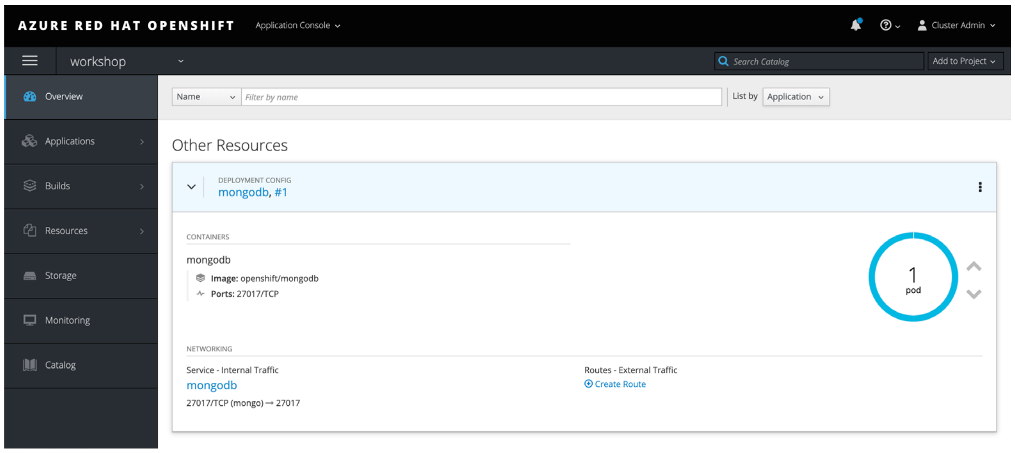
You can retrieve a list of templates using the command below. The templates are preinstalled in the openshift namespace.

```
oc get templates -n openshift
```

Create a MongoDB deployment using the mongodb-persistent template. You're passing in the values to be replaced (username, password and database) which generates a YAML/JSON file. You then pipe it to the oc create command.

```
oc process openshift//mongodb-persistent \
    -p MONGODB_USER=ratingsuser \
    -p MONGODB_PASSWORD=ratingspassword \
    -p MONGODB_DATABASE=ratingsdb \
    -p MONGODB_ADMIN_PASSWORD=ratingspassword | oc create -f -
```

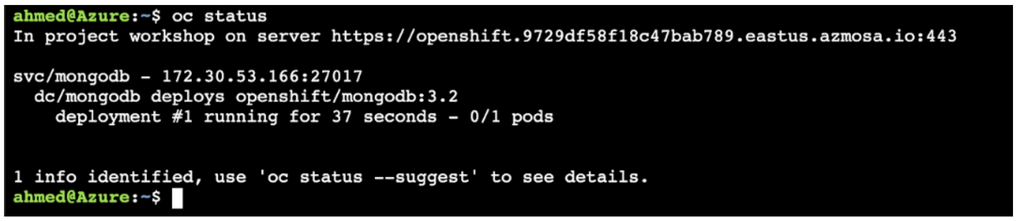
If you now head back to the web console, you should see a new deployment for MongoDB.



Verify the MongoDB pod was created successfully

Run the oc status command to view the status of the new application and verify if the deployment of the mongoDB template was successful.

```
oc status
```

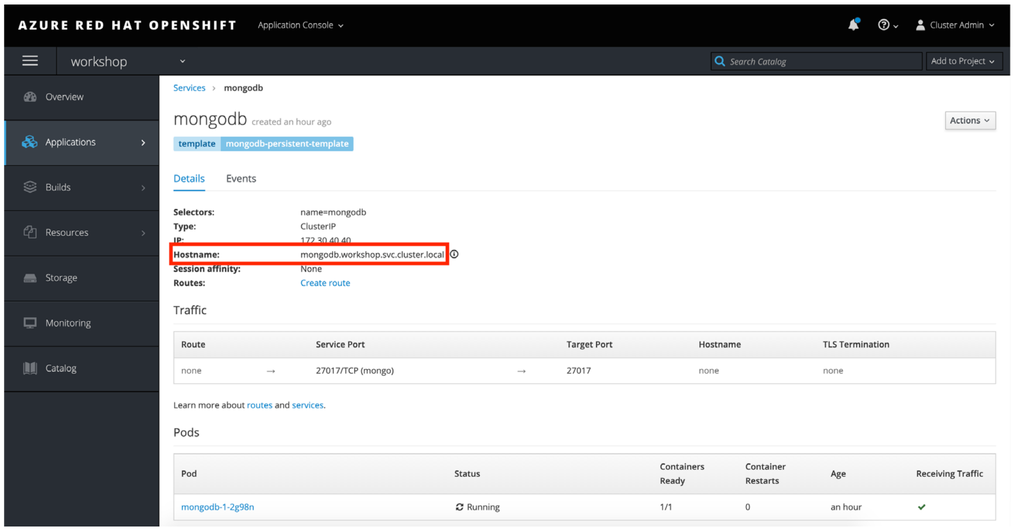


```
ahmed@Azure:~$ oc status
In project workshop on server https://openshift.9729df58f18c47bab789.eastus.azmosa.io:443

svc/mongodb - 172.30.53.166:27017
  dc/mongodb deploys openshift/mongodb:3.2
    deployment #1 running for 37 seconds - 0/1 pods


1 info identified, use 'oc status --suggest' to see details.
ahmed@Azure:~$
```

Retrieve the MongoDB service hostname

The service will be accessible at the following hostname: mongodb.workshop.svc.cluster.local which is formed of [service name].[project name].svc.cluster.local. This resolves only within the cluster.

You can also retrieve this from the web console. You'll need this hostname to configure the rating-api.



mongodb created an hour ago

template mongodb-persistent-template

Selectors: name=mongodb
Type: ClusterIP
IP: 172.30.40.40
Hostname: mongodb.workshop.svc.cluster.local
Session affinity: None
Routes: Create route

Traffic

Route		Service Port		Target Port	Hostname	TLS Termination
none	→	27017/TCP (mongo)	→	27017	none	none

Learn more about routes and services.

Pods

Pod	Status	Containers Ready	Container Restarts	Age	Receiving Traffic
mongodb-1-2g98n	Running	1/1	0	an hour	✓

Deploy the ratings-api service

The rating-api is a NodeJS application that connects to m\MongoDB to retrieve and rate items. Below are some of the details that you'll need to deploy this.

- rating-api on GitHub: https://github.com/microsoft/rating-api
- The container exposes port 8080
- MongoDB connection is configured using an environment variable called MONGODB_URI

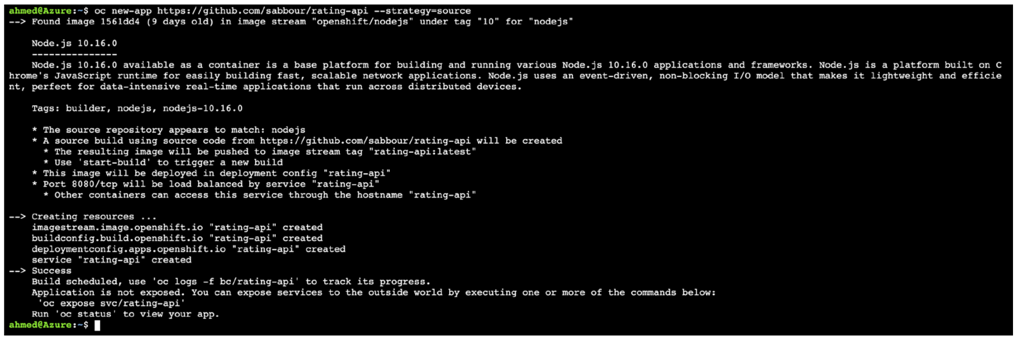
Use the OpenShift CLI to deploy the rating-api

Note that to be able to setup CI/CD webhooks, you'll need to fork the application into your personal GitHub repository first. After that, you're going to be using source-to-image (S2I) as a build strategy.

Create a new application in the project by pointing to your GitHub fork of the rating-api app.

```
oc new-app https://github.com/<your GitHub username>/rating-api
--strategy=source
```

OpenShift should now pull the source code, detect that this is a NodeJS application then use S2I to build a container image and push it to the built-in container registry. OpenShift is also going to deploy the application using a deployment config and create a service.

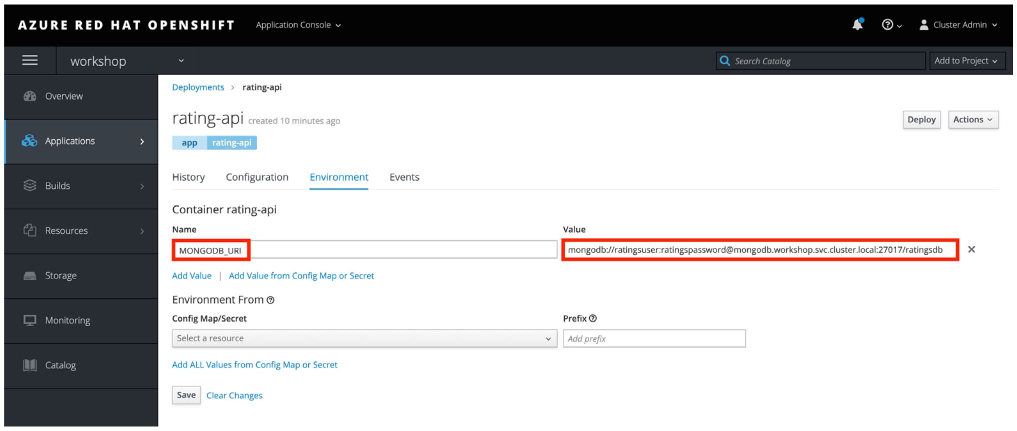


```
ahmed@Azure:~$ oc new-app https://github.com/sabbour/rating-api --strategy=source
--> Found image 1561dd4 (9 days old) in image stream "openshift/nodejs" under tag "10" for "nodejs"

    Node.js 10.16.0
    ---------------
    Node.js 10.16.0 available as a container is a base platform for building and running various Node.js 10.16.0 applications and frameworks. Node.js is a platform built on C
hrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficie
nt, perfect for data-intensive real-time applications that run across distributed devices.

    Tags: builder, nodejs, nodejs-10.16.0

    * The source repository appears to match: nodejs
    * A source build using source code from https://github.com/sabbour/rating-api will be created
      * The resulting image will be pushed to image stream tag "rating-api:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "rating-api"
    * Port 8080/tcp will be load balanced by service "rating-api"
      * Other containers can access this service through the hostname "rating-api"

--> Creating resources ...
    imagestream.image.openshift.io "rating-api" created
    buildconfig.build.openshift.io "rating-api" created
    deploymentconfig.apps.openshift.io "rating-api" created
    service "rating-api" created
--> Success
    Build scheduled, use 'oc logs -f bc/rating-api' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
     'oc expose svc/rating-api'
    Run 'oc status' to view your app.
ahmed@Azure:~$
```

Configure the required environment variables

The rating-api application expects to find the MongoDB connection information in an environment variable called MONGODB_URI. This URI should look like *mongodb://[username]:[password]@[endpoint]:27017/ratingsdb*. You'll need to replace the *[username]* and *[password]* with the ones you used when creating the database. You'll also need to replace the *[endpoint]* with the hostname acquired in the previous section.

You can accomplish this task using the OpenShift CLI or using the Web Console, for this one you'll edit the deployment configuration for the rating-api application on the Web Console to add the environment variable. Make sure to hit **Save** when done.



Container rating-api

Name	Value
MONGODB_URI	mongodb://ratingsuser:ratingspassword@mongodb.workshop.svc.cluster.local:27017/ratingsdb

Config Map/Secret	Prefix
Select a resource	Add prefix

Deploy the ratings-web frontend using S2I strategy

The rating-web is a NodeJS application that connects to the rating-api. Below are some of the details that you'll need to deploy this.

- rating-web on GitHub
- The container exposes port 8080
- The web app connects to the API over the internal cluster DNS, using a proxy through an environment variable named API

Create a route for the ratings-web frontend

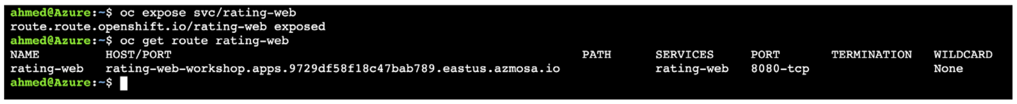
Expose the service.

```
oc expose svc/rating-web
```

Find out the created route hostname

```
oc get route rating-web
```

You should get a response similar to the below.



```
ahmed@Azure:~$ oc expose svc/rating-web
route.route.openshift.io/rating-web exposed
ahmed@Azure:~$ oc get route rating-web
NAME         HOST/PORT                                                   PATH      SERVICES     PORT       TERMINATION   WILDCARD
rating-web   rating-web-workshop.apps.9729df58f18c47bab789.eastus.azmosa.io        rating-web   8080-tcp                 None
ahmed@Azure:~$
```

Notice the fully qualified domain name (FQDN) is comprised of the application name and project name by default. The remainder of the FQDN, the subdomain, is your Azure Red Hat OpenShift cluster specific apps subdomain.

Scaling the application and the cluster

You can scale the number of application nodes in the cluster using the Azure CLI. Run the below on the Azure Cloud Shell to scale your cluster to 5 application nodes. Replace <*cluster name*> and <*resource group name*> with your applicable values. After a few minutes, *az openshift scale* will complete successfully and return a JSON document containing the scaled cluster details.

```
az openshift scale --name <cluster name> --resource-group <resource
    group name> --compute-count 5
```

After the cluster has scaled successfully. You can run the following command to verify the number of application nodes.

```
az openshift show --name <cluster name> --resource-group <resource
    group name> --query “agentPoolProfiles”[0]
```

Following is a sample output. You can notice that the value of count for agentPoolProfiles has been scaled to 5.

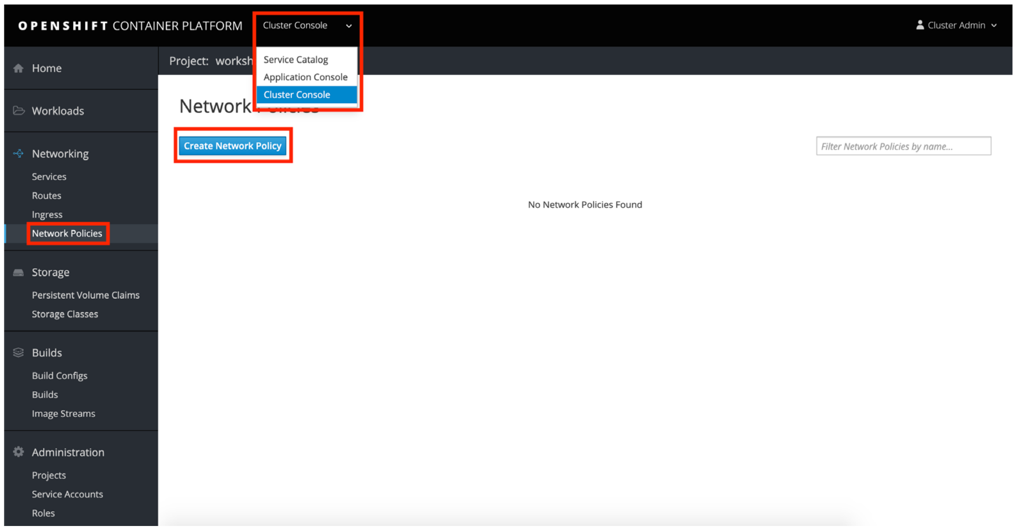
```
{
    “count”: 5,
    “name”: “compute”,
    “osType”: “Linux”,
    “role”: “compute”,
    “subnetCidr”: “10.0.0.0/24”,
    “vmSize”: “Standard_D4s_v3”
}
```

Controlling networking using networking policies

Now that you have the application working, it is time to apply some security hardening. You'll use network policies to restrict communication to the rating-api.

Switch to the Cluster Console

Switch to the **Cluster Console** page. Switch to project **workshop**. Click **Create Network Policy**.

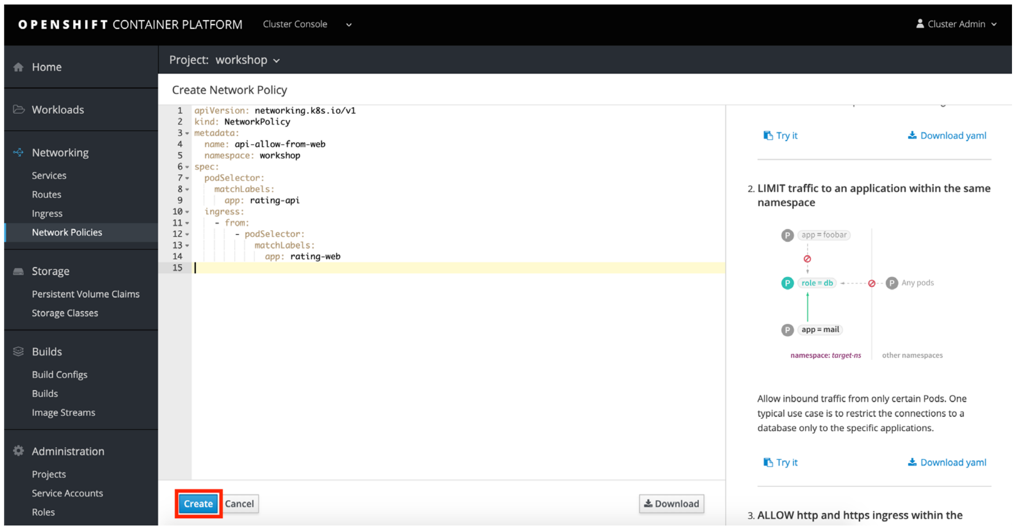


Create Network Policy

You will create a policy that applies to any pod matching the app=rating-api label. The policy will allow ingress only from pods matching the app=rating-web label.

Use the YAML below in the editor, and make sure you're targeting the **workshop** project.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-allow-from-web
  namespace: workshop
spec:
  podSelector:
    matchLabels:
      app: rating-api
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: rating-web
```



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-allow-from-web
  namespace: workshop
spec:
  podSelector:
    matchLabels:
      app: rating-api
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: rating-web
```

2. **LIMIT traffic to an application within the same namespace**

Allow inbound traffic from only certain Pods. One typical use case is to restrict the connections to a database only to the specific applications.

3. **ALLOW http and https ingress within the**

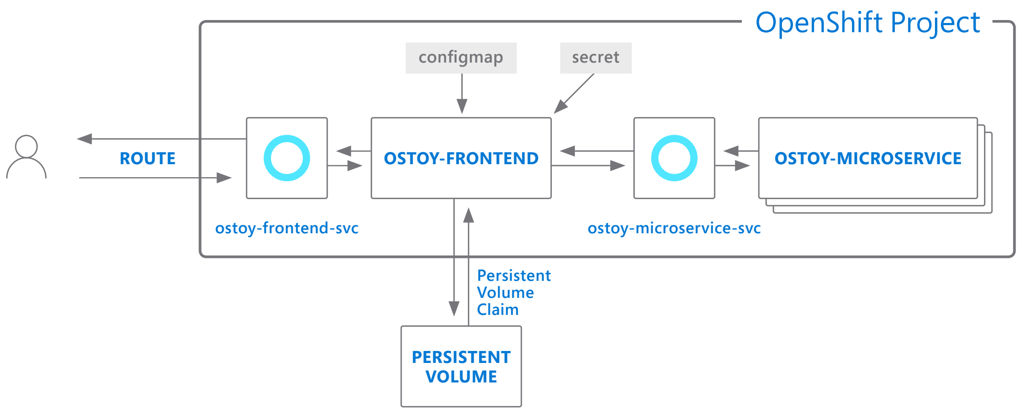
Click **Create**.

USING AN APP TO BECOME FAMILIAR WITH OPENSHIFT AND KUBERNETES

Application Overview

OSToy is a simple Node.js application that we will deploy to Azure Red Hat OpenShift. It is used to help us explore the functionality of Kubernetes. This application has a user interface which you can:

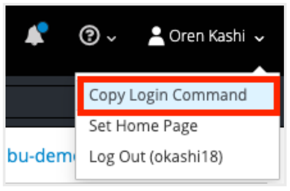
- write messages to the log (stdout / stderr)
- intentionally crash the application to view self-healing
- toggle a liveness probe and monitor OpenShift behavior
- read config maps, secrets, and env variables
- if connected to shared storage, read and write files
- check network connectivity, intra-cluster DNS, and intra-communication with an included microservice



Deploy the OSToy application

Retrieve login command

If not logged in via the CLI, click on the dropdown arrow next to your name in the top-right and select *Copy Login Command*.



Then go to your terminal and paste that command and press enter. You will see a similar confirmation message if you successfully logged in.

```
$ oc login https://openshift.abcd1234.eastus.azmosa.io
    --token=hUXXXXXX
Logged into "https://openshift.abcd1234.eastus.azmosa.io:443" as
    "okashi" using the token provided.
```

You have access to the following projects and can switch between them with 'oc project <projectname>':

```
    aro-demo
*   aro-shifty
    ...
```

Create new project

Create a new project called "OSToy" in your cluster.

Use the following command

```
    oc new-project ostoy
```

You should receive the following response:

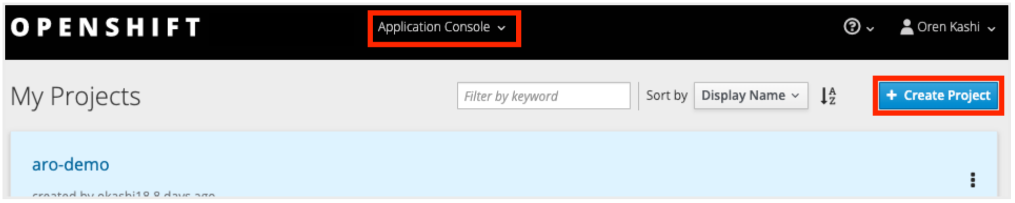
```
    $ oc new-project ostoy
    Now using project "ostoy" on server
    "https://openshift.abcd1234.eastus.azmosa.io:443".
```

You can add applications to this project with the 'new-app' command. For example, try:

```
    oc new-app
    centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git
```

to build a new example application **in** Ruby.

You can also create this new project using the web UI by selecting "Application Console" at the top and then clicking the "+Create Project" button on the right.



Download YAML configuration

Download the Kubernetes deployment object yamls from the following locations to your Azure Cloud Shell, in a directory of your choosing (just remember where you placed them for the next step).

Feel free to open them up and take a look at what we will be deploying. For simplicity of this lab we have placed all the Kubernetes objects we are deploying in one "all-in-one" yaml file. Though in reality there are benefits to separating these out into individual yaml files.

ostoy-fe-deployment.yaml
ostoy-microservice-deployment.yaml

Deploy backend microservice

The microservice application serves internal web requests and returns a JSON object containing the current hostname and a randomly generated color string.

In your command line, deploy the microservice using the following command:

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

You should see the following response:

```
$ oc apply -f ostoy-microservice-deployment.yaml
deployment.apps/ostoy-microservice created
service/ostoy-microservice-svc created
```

Deploy the front-end service

The frontend deployment contains the node.js frontend for our application along with a few other Kubernetes objects to illustrate examples.

If you open the *ostoy-fe-deployment.yaml* you will see we are defining:

- Persistent Volume Claim
- Deployment Object
- Service
- Route
- Configmaps
- Secrets

In your command line deploy the frontend along with creating all objects mentioned above by entering:

```
    oc apply -f ostoy-fe-deployment.yaml
```

You should see all objects created successfully

```
$ oc apply -f ostoy-fe-deployment.yaml
persistentvolumeclaim/ostoy-pvc created
deployment.apps/ostoy-frontend created
service/ostoy-frontend-svc created
route.route.openshift.io/ostoy-route created
configmap/ostoy-configmap-env created
secret/ostoy-secret-env created
configmap/ostoy-configmap-files created
secret/ostoy-secret created
```

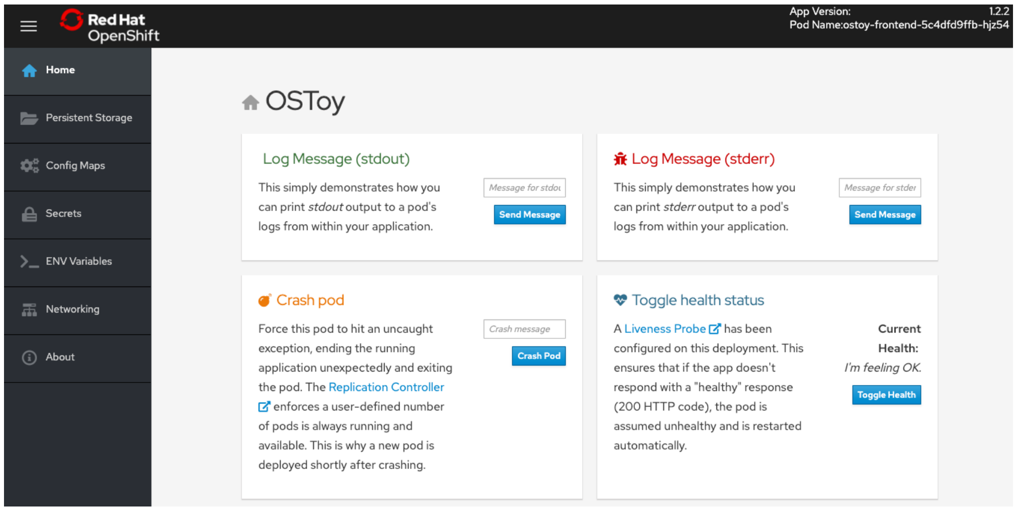
Get route

Get the route so that we can access the application via oc *get route*

You should see the following response:

```
NAME         HOST/PORT                                     PATH               SERVICES   PORT   TERMINATION   WILDCARD
ostoy-route  ostoy-route-ostoy.apps.abcd1234.eastus.azmosa.is   ostoy-frontend-svc  <all>          None
```

Copy *ostoy-route-ostoy.apps.abcd1234.eastus.azmosa.io* above and paste it into your browser and press enter. You should see the homepage of our application.



This simply demonstrates how you can print *stdout* output to a pod's logs from within your application.

This simply demonstrates how you can print *stderr* output to a pod's logs from within your application.

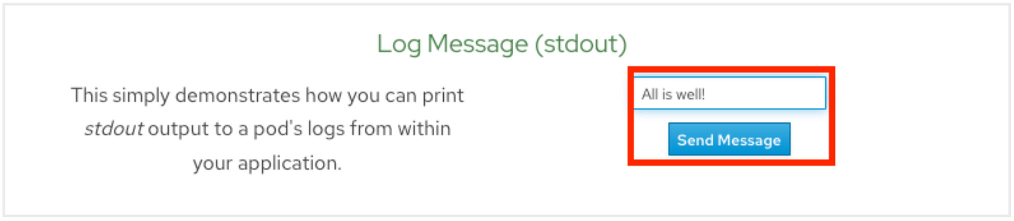
Force this pod to hit an uncaught exception, ending the running application unexpectedly and exiting the pod. The Replication Controller enforces a user-defined number of pods is always running and available. This is why a new pod is deployed shortly after crashing.

A Liveness Probe has been configured on this deployment. This ensures that if the app doesn't respond with a "healthy" response (200 HTTP code), the pod is assumed unhealthy and is restarted automatically.

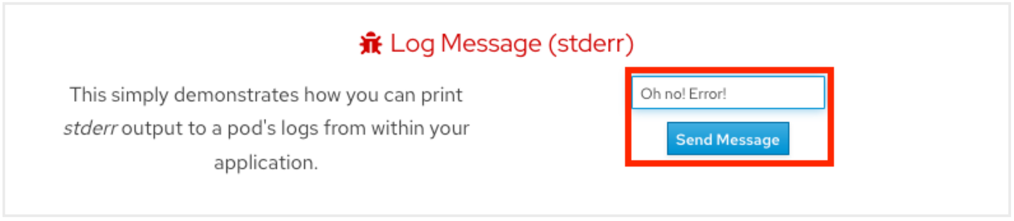
Explore Logging

Assuming you can access the application via the Route provided and are still logged into the CLI (please go back to part 2 if you need to do any of those), we'll start to use this application. As stated earlier, this application will allow you to "push the buttons" of OpenShift and see how it works.

Click on the Home menu item and then click in the message box for "Log Message (stdout)" and write any message you want to output to the stdout stream. You can try "**All is well!**". Then click "Send Message".


Log Message (stdout)
This simply demonstrates how you can print stdout output to a pod's logs from within your application.
All is well!
Send Message

Click in the message box for "Log Message (stderr)" and write any message you want to output to the stderr stream. You can try "**Oh no! Error!**". Then click "Send Message".


Log Message (stderr)
This simply demonstrates how you can print stderr output to a pod's logs from within your application.
Oh no! Error!
Send Message

Go to the CLI and enter the following command to retrieve the name of your frontend pod which we will use to view the pod logs:

```
$ oc get pods -o name
pod/ostoy-frontend-679cb85695-5cn7x
pod/ostoy-microservice-86b4c6f559-p594d
```

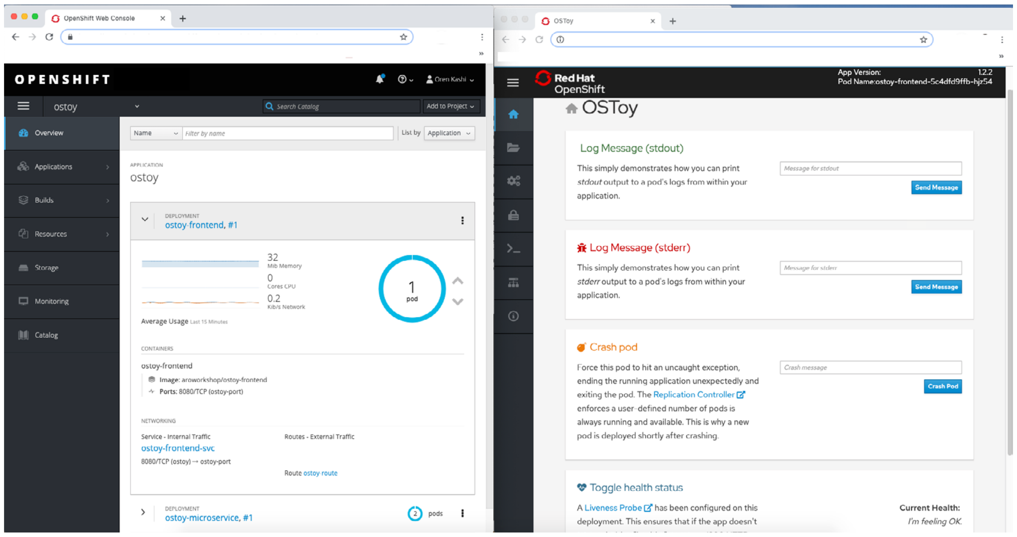
So the pod name in this case is **ostoy-frontend-679cb85695-5cn7x**. Then run oc logs ostoy-frontend-679cb85695-5cn7x and you should see your messages:

```
$ oc logs ostoy-frontend-679cb85695-5cn7x
[...]
ostoy-frontend-679cb85695-5cn7x: server starting on port 8080
Redirecting to /home
stdout: All is well!
stderr: Oh no! Error!
```

You should see both the *stdout* and *stderr* messages.

Health Checks

It would be best to prepare by splitting your screen between the OpenShift Web UI and the OSToy application so that you can see the results of our actions immediately.



APPLICATION
ostoy

DEPLOYMENT
ostoy-frontend, #1

32 Mib Memory
0 Cores CPU
0.2 Kib/s Network
Average Usage Last 15 Minutes
1 pod

CONTAINERS
ostoy-frontend
Image: aroworkshop/ostoy-frontend
Ports: 8080/TCP (ostoy-port)

NETWORKING
Service - Internal Traffic
ostoy-frontend-svc
8080/TCP (ostoy) → ostoy-port
Routes - External Traffic
Route ostoy route

DEPLOYMENT
ostoy-microservice, #1
2 pods

App Version: 1.2.2
Pod Name:ostoy-frontend-5c4dfd9ffb-hjz54

OSToy

Log Message (stdout)
This simply demonstrates how you can print *stdout* output to a pod's logs from within your application.

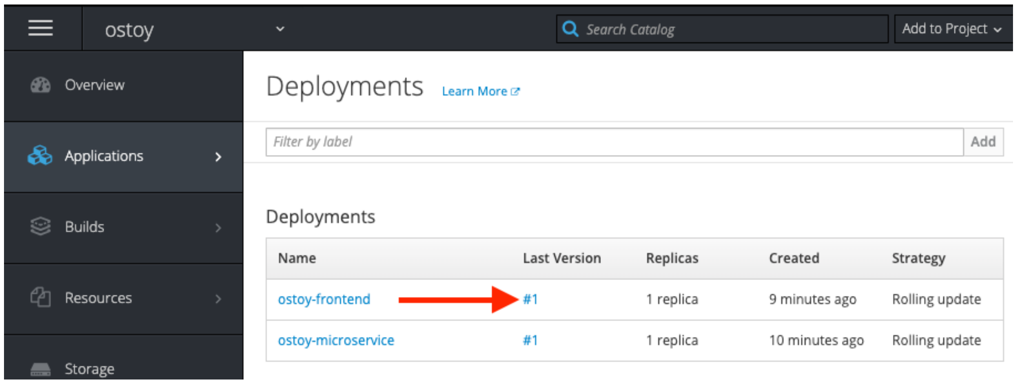
Log Message (stderr)
This simply demonstrates how you can print *stderr* output to a pod's logs from within your application.

Crash pod
Force this pod to hit an uncaught exception, ending the running application unexpectedly and exiting the pod. The Replication Controller enforces a user-defined number of pods is always running and available. This is why a new pod is deployed shortly after crashing.

Toggle health status
A Liveness Probe has been configured on this deployment. This ensures that if the app doesn't
Current Health:
I'm feeling OK

If your screen is too small or that just won't work, then open the OSToy application in another tab so you can quickly switch to the OpenShift Web Console once you click the button. To get to this deployment in the OpenShift Web Console go to:

Applications > Deployments > click the number in the "Last Version" column for the "ostoy-frontend" row

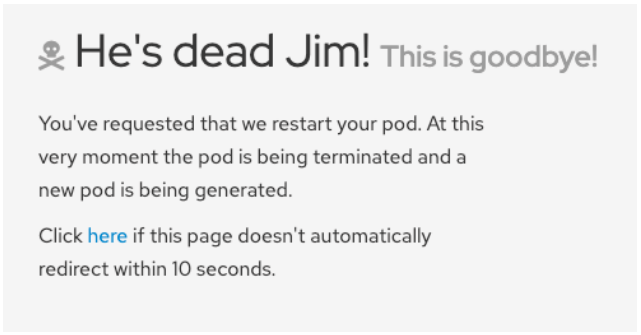


Deployments

Deployments

Name	Last Version	Replicas	Created	Strategy
ostoy-frontend	#1	1 replica	9 minutes ago	Rolling update
ostoy-microservice	#1	1 replica	10 minutes ago	Rolling update

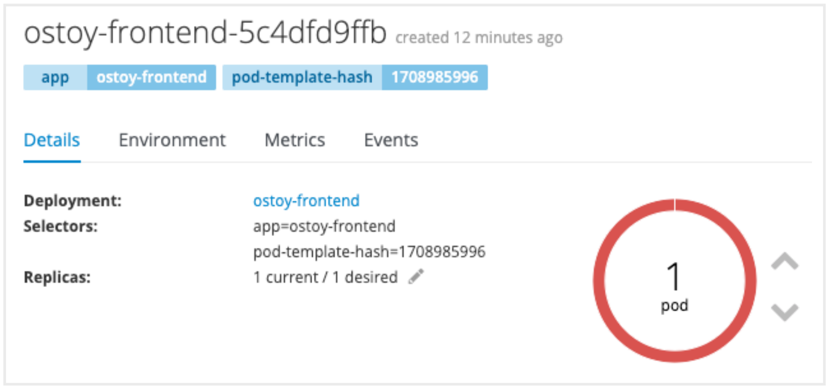
Go to the OSToy app, click on Home in the left menu, and enter a message in the "Crash Pod" tile (ie: "This is goodbye!") and press the "Crash Pod" button. This will cause the pod to crash and Kubernetes should restart the pod. After you press the button you will see:


He's dead Jim! This is goodbye!

You've requested that we restart your pod. At this very moment the pod is being terminated and a new pod is being generated.

Click here if this page doesn't automatically redirect within 10 seconds.

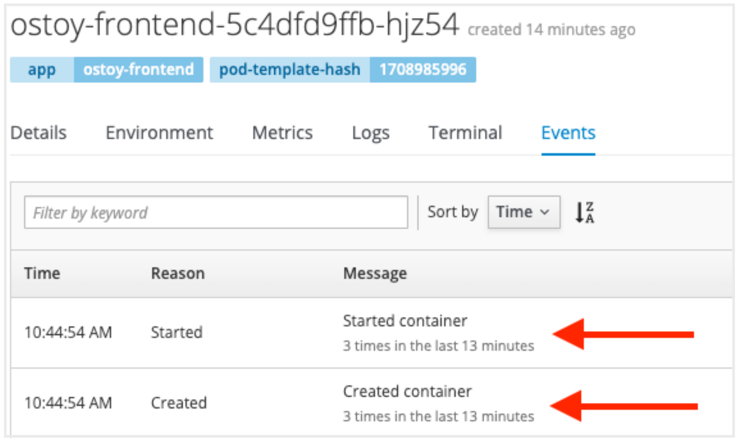
Quickly switch to the Deployment screen. You will see that the pod is red, meaning it is down but should quickly come back up and show blue.


ostoy-frontend-5c4dfd9ffb created 12 minutes ago

app ostoy-frontend pod-template-hash 1708985996

Deployment: ostoy-frontend
Selectors: app=ostoy-frontend
pod-template-hash=1708985996
Replicas: 1 current / 1 desired

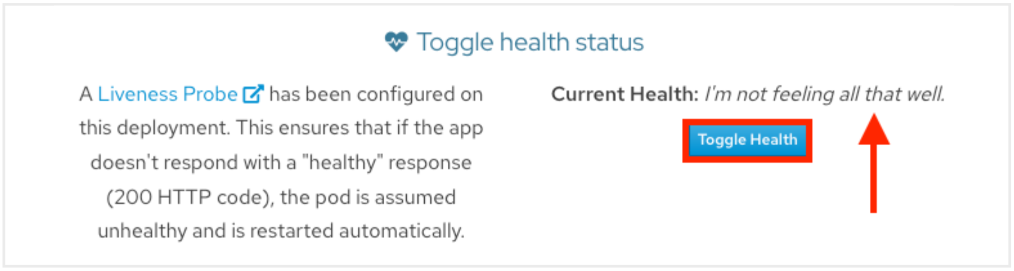
You can also check in the pod events and further verify that the container has crashed and been restarted.


ostoy-frontend-5c4dfd9ffb-hjz54 created 14 minutes ago

app ostoy-frontend pod-template-hash 1708985996

Time	Reason	Message
10:44:54 AM	Started	Started container 3 times in the last 13 minutes
10:44:54 AM	Created	Created container 3 times in the last 13 minutes

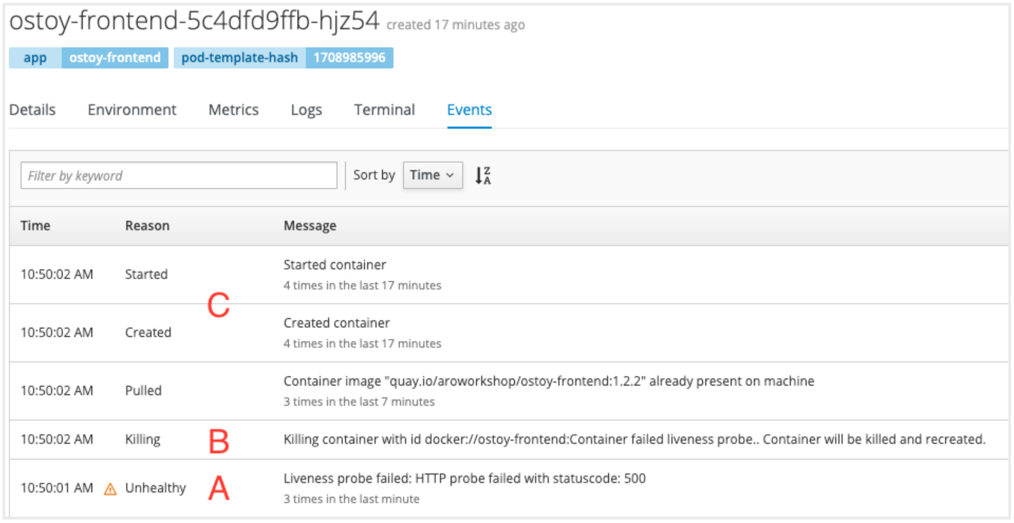
Keep the page from the pod events still open from step 4. Then in the OSToy app click on the “Toggle Health” button, in the “Toggle Health Status” tile. You will see the “Current Health” switch to “I’m not feeling all that well”.



A Liveness Probe has been configured on this deployment. This ensures that if the app doesn't respond with a "healthy" response (200 HTTP code), the pod is assumed unhealthy and is restarted automatically.

Current Health: *I'm not feeling all that well.*

This will cause the app to stop responding with a “200 HTTP code”. After 3 such consecutive failures (“A”), Kubernetes will kill the pod (“B”) and restart it (“C”). Quickly switch back to the pod events tab and you will see that the liveness probe failed and the pod as being restarted.



Time	Reason	Message
10:50:02 AM	Started	Started container 4 times in the last 17 minutes
10:50:02 AM	Created	Created container 4 times in the last 17 minutes
10:50:02 AM	Pulled	Container image "quay.io/aroworkshop/ostoy-frontend:1.2.2" already present on machine 3 times in the last 7 minutes
10:50:02 AM	Killing	Killing container with id docker://ostoy-frontend:Container failed liveness probe.. Container will be killed and recreated.
10:50:01 AM	⚠ Unhealthy	Liveness probe failed: HTTP probe failed with statuscode: 500 3 times in the last minute

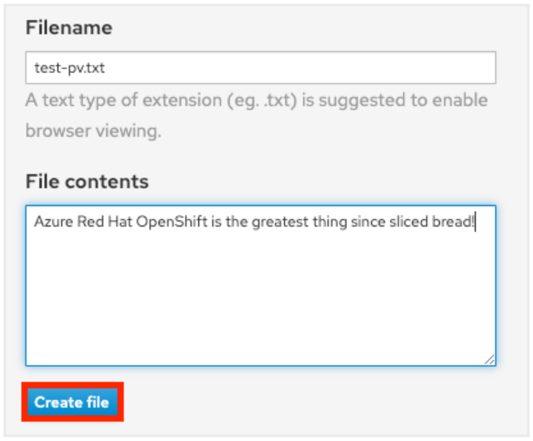
Persistent Storage

Inside the OpenShift web UI click on Storage in the left menu. You will then see a list of all persistent volume claims that our application has made. In this case there is just one called "ostoy-pvc". You will also see other pertinent information such as whether it is bound or not, size, access mode and age.

In this case the mode is RWO (Read-Write-Once) which means that the volume can only be mounted to one node, but the pod(s) can both read and write to that volume. The default in Azure Red Hat OpenShift is for Persistent Volumes to be backed by Azure Disk, but it is possible to chose Azure Files so that you can use the RWX (Read-Write-Many) access mode. (See here for more info on access modes)

In the OSToy app click on Persistent Storage in the left menu. In the "Filename" area enter a filename for the file you will create. (ie: "test-pv.txt")

Underneath that, in the "File Contents" box, enter text to be stored in the file. (ie: "Azure Red Hat OpenShift is the greatest thing since sliced bread!" or "test" :)).
Then click "Create file".


Filename

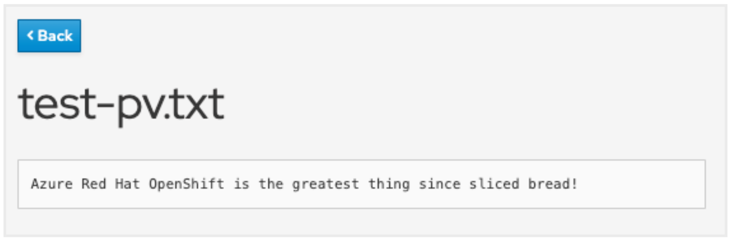
test-pv.txt

A text type of extension (eg. .txt) is suggested to enable browser viewing.

File contents

Azure Red Hat OpenShift is the greatest thing since sliced bread!

You will then see the file you created appear above under "Existing files". Click on the file and you will see the filename and the contents you entered.



test-pv.txt

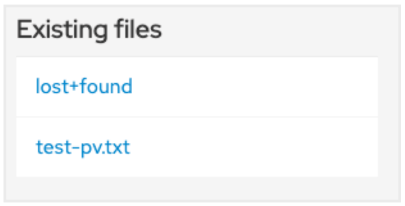
```
Azure Red Hat OpenShift is the greatest thing since sliced bread!
```

We now want to kill the pod and ensure that the new pod that spins up will be able to see the file we created. Exactly like we did in the previous section. Click on *Home* in the left menu.

Click on the “Crash pod” button. (You can enter a message if you’d like).

Click on *Persistent Storage* in the left menu

You will see the file you created is still there and you can open it to view its contents to confirm.


Existing files

lost+found

test-pv.txt

Now let’s confirm that it’s actually there by using the CLI and checking if it is available to the container. If you remember, we mounted the directory /var/demo-files to our PVC. So get the name of your frontend pod

```
oc get pods
```

then get an SSH session into the container

```
oc rsh <podname>
```

then

```
cd /var/demo-files
```

if you enter ls you can see all the files you created. Next, let’s open the file we created and see the contents

```
cat test-pv.txt
```

You should see the text you entered in the UI.

```
$ oc get pods
NAME                                 READY   STATUS    RESTARTS   AGE
ostoy-frontend-5fc8d486dc-wsw24      1/1     Running   0          18m
ostoy-microservice-6cf764974f-hx4qm  1/1     Running   0          18m

$ oc rsh ostoy-frontend-5fc8d486dc-wsw24
/ $ cd /var/demo_files/

/var/demo_files $ ls
lost+found test-pv.txt

/var/demo_files $ cat test-pv.txt
Azure Red Hat OpenShift is the greatest thing since sliced bread!
```

Then exit the SSH session by typing exit. You will then be in your CLI.

Configuration - ConfigMaps, Secrets, Environment Variables

In this section we'll take a look at how OSToy can be configured using ConfigMaps, Secrets, and Environment Variables. This section won't go into details explaining each (the links above are for that), but will show you how they are exposed to the application.

Configuration using ConfigMaps

ConfigMaps allow you to decouple configuration artifacts from container image content to keep containerized applications portable.

Click on *Config Maps* in the left menu.

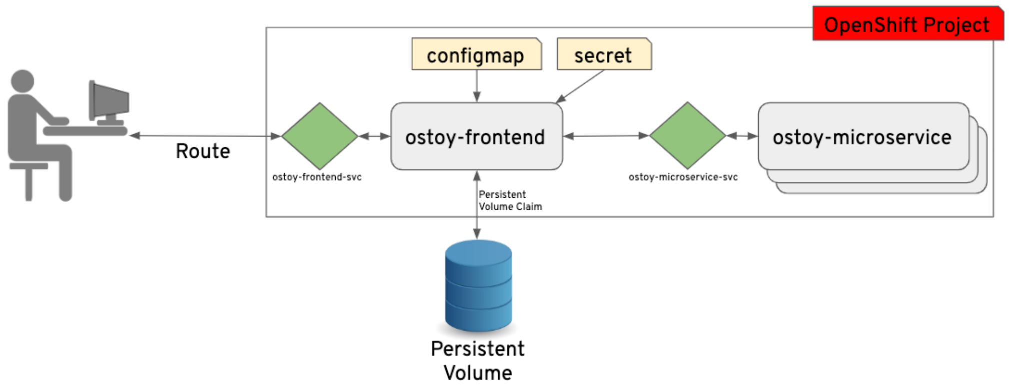
This will display the contents of the configmap available to the OSToy application. We defined this in the ostoy-fe-deployment.yaml here:

```
kind: ConfigMap
apiVersion: v1
metadata:
   name: ostoy-configmap-files
data:
   config.json: '{ "default": "123" }'
```

Networking and Scaling

In this section we'll see how OSToy uses intra-cluster networking to separate functions by using microservices and visualize the scaling of pods.

Let's review how this application is set up...



As can be seen in the image above, we have defined at least 2 separate pods, each with its own service. One is the frontend web application (with a service and a publicly accessible route) and the other is the backend microservice with a service object created so that the frontend pod can communicate with the microservice (across the pods if more than one). Therefore this microservice is not accessible from outside this cluster, nor from other namespaces/projects (due to Azure Red Hat OpenShift's network policy, **ovs-networkpolicy**). The sole purpose of this microservice is to serve internal web requests and return a JSON object containing the current hostname and a randomly generated color string. This color string is used to display a box with that color displayed in the tile (titled "Intra-cluster Communication").

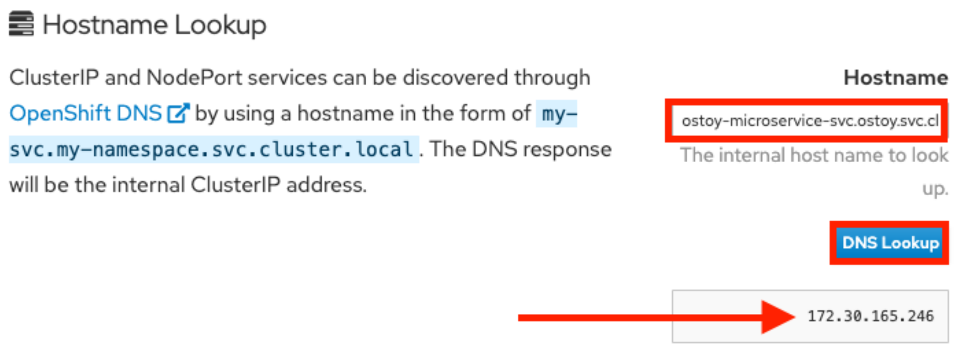
Networking

Click on Networking in the left menu. Review the networking configuration.

The right tile titled "Hostname Lookup" illustrates how the service name created for a pod can be used to translate into an internal ClusterIP address. Enter the name of the microservice following the format of *my-svc.my-namespace.svc.cluster.local* which we created in our *ostoy-microservice.yaml* which can be seen here:

```
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
```

In this case we will enter: *ostoy-microservice-svc.ostoy.svc.cluster.local*
We will see an IP address returned. In our example it is *172.30.165.246*. This is the intra-cluster IP address; only accessible from within the cluster.


Hostname Lookup

ClusterIP and NodePort services can be discovered through OpenShift DNS by using a hostname in the form of `my-svc.my-namespace.svc.cluster.local`. The DNS response will be the internal ClusterIP address.

Hostname: ostoy-microservice-svc.ostoy.svc.cl

The internal host name to look up.

```
172.30.165.246
```

Scaling

OpenShift allows one to scale up/down the number of pods for each part of an application as needed. This can be accomplished via changing our *replicaset/deployment* definition (declarative), by the command line (imperative), or via the web UI (imperative). In our deployment definition (part of our ostoy-fe-deployment.yaml) we stated that we only want one pod for our microservice to start with. This means that the Kubernetes Replication Controller will always strive to keep one pod alive.

CONCLUSION

Running Kubernetes alone may still allow you to achieve the level of cluster management you are looking for, but it comes with a price.

When your development and operations teams spend most of their working hours dealing with provisioning, setup, maintenance, and overseeing your clusters and CI/CD pipeline, they're not able to dedicate their valuable time towards what they are best at – keeping your apps at the cutting edge.

As we have learned from this guide, Azure Red Hat OpenShift lets you deploy fully managed Red Hat OpenShift clusters without worrying about building or managing the infrastructure required to run it. We've seen that running Kubernetes alone comes with a few caveats, mainly in relation with the extra hands-on attention required with tasks that could be automated with Azure Red Hat OpenShift.

When you're deciding which cluster management strategy to choose for your organization, consider the pros and cons that you'll be getting with a Kubernetes type of platform versus Azure Red Hat OpenShift, which is built on the Kubernetes framework and offers you a bundle of extra out-of-the-box benefits.

To learn more about your Azure Red Hat OpenShift, visit the product page or check out our documentation section. You can also go through a hands-on workshop, and register to watch a webinar at your convenience.