Zero to Cloud-Native with IBM Cloud

Part 1: Introduction and Overview

Kevin Collins
Technical Sales Leader
IBM Cloud Enterprise Containers – Americas

Introduction

With over 200 services available in IBM Cloud Catalog you can develop virtual any cloud native application that you can dream of. But how can you best get started. The purpose of this blog series 'Zero to Cloud-Native with IBM Cloud' is to showcase how to get started in developing with the IBM Cloud Catalog. This multi-part series will cover topics such as architecture design and how to select the best services available in the IBM Cloud Catalog based on your architecture and application, how to combine services together and then how to implement day two operations while you are developing and also after you have deployed your application.

You can find links to all the blog entries, source code and videos at this GitHub repository: https://github.com/kmcolli/zero-to-cloud-native-ibm-cloud

The series will be short blogs and accompanying videos describing a topic while going through the cloud-native development lifecycle. Initially the series will go through the following:

1. Introduction – overview and introduction to developing a cloud native application with IBM Cloud.
2. Microservices Design – design considerations for developing a cloud-native application including patterns to follow when developing real time and long duration APIs.
3. Network Configuration – network architecture and security design considerations for a cloud-native application.
4. RedHat OpenShift Cluster Configuration – deployment considerations and how to design a cluster for development, staging and production environments.
5. Microservices Messaging – how to use a managed service as a messaging broker to de-couple microservices and improve performance.
6. Cloud Native Database Considerations – examples of selecting a purpose built database as a service to optimize the application architecture to support different data types.
7. DevOps – one of the first and most important things you can do when developing a cloud-native application is establishing good dev ops processes. This section will go through setting up CI/CD toolchains with both Classic and Tekton pipelines and detail lessons learned.
8. Logging – how to leverage logging while developing your cloud-native application and how to leverage logging to support the application while running in a production environment.

9. Monitoring – best practices to design and implement a monitoring dashboard for a cloud-native application.
10. Tuning / Capacity Planning – how to tune and scale your application to make the best use of your cloud resources in a cost effective manner.
11. Clone and Re-Instate Deep Dive – how to leverage IBM Cloud to clone and re-instate your cluster and namespaces.

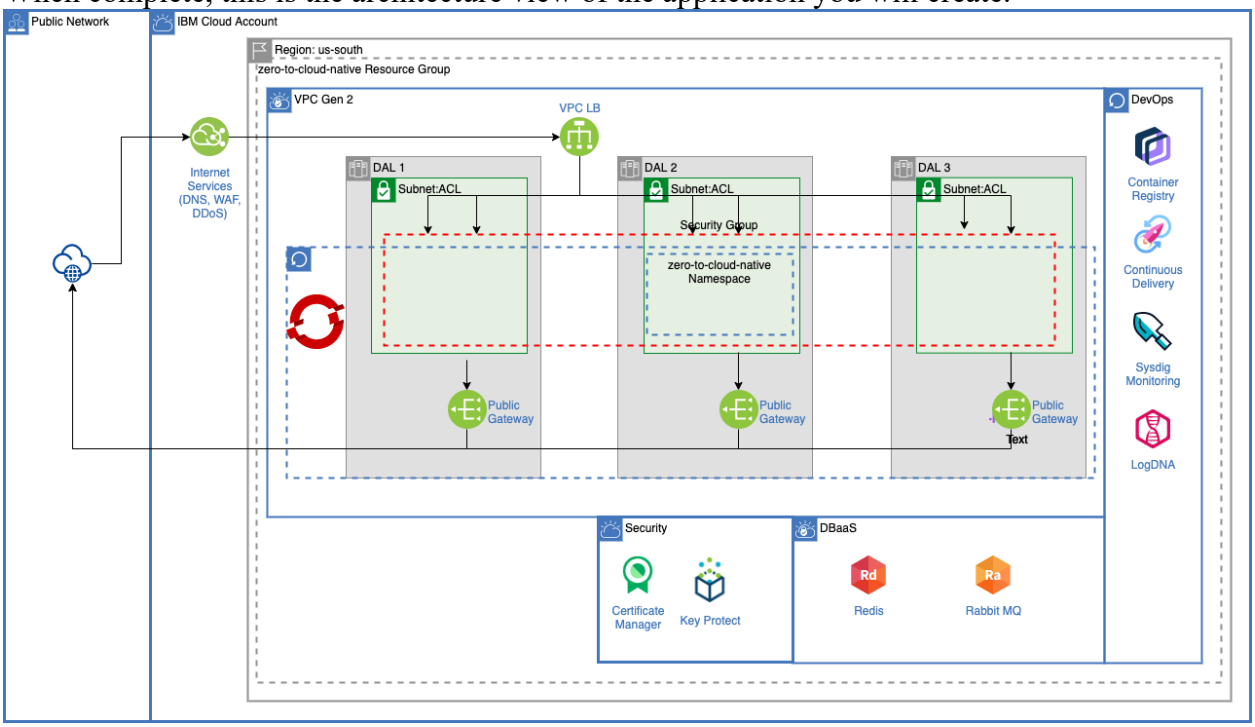
Tutorial Application

The best what to learn from my perspective is getting hands on experience. The accompanying tutorial will go through the entire cloud-native development lifecycle. I will walk through step by step to create an entire cloud native application on IBM Cloud leveraging the IBM Cloud Catalog.

The tutorials will go through the following topics:

- Microservices design
- Creating a new isolated resource group
- Creating a custom domain for your application.
- Setting up and configuring a virtual private cloud including configuring subnets, access control lists and security groups.
- Configuration of IBM Cloud Internet Services to secure and provide DNS services for a custom domain
- Leveraging certificate manager to create and managed domain certificates
- Using Key Protect to manage your own certificates across IBM Cloud services
- Provisioning and configuring a managed OpenShift Cluster in a multi-zone region
- Provisioning and configuring Rabbit MQ for messaging
- Provisioning and configuring IBM Cloud Databases for a data layer
- Creating a classic CI/CD pipeline with integration with an enterprise git repository
- Creating a tekton CI/CD pipeline with integration with an enterprise git repository
- Deploying and running the application on OpenShift
- Leveraging IBM Log Analysis with LogDNA to provide logging and troubleshoot both from a development and support perspective.
- Leveraging IBM Systems Monitoring with SysDig to monitor IBM Cloud Platform components, OpenShift cluster and the application itself.
- Tuning your cluster to optimize availability and resources.
- New functionality from IBM Cloud to clone and re-instate cluster namespaces

When complete, this is the architecture view of the application you will create.



The tutorial application will comprise of three APIs:

1) **getOCPVersions** – will return a list of OpenShift versions supported by IBM Cloud Managed OpenShift
2) **getOCPToken** – will return an OCP token to login into your cluster.
3) **enableNodeSSH** – will enable your IBM Cloud Managed OpenShift worker nodes to be SSH'd into.

To support these APIs, the following microservices, all written in Python, will be created.

- **frontend-web.py** – HTML front end to invoke the three public facing APIs
- **frontend-api.py** – API interface to call the public facing APIs
- **getOCPVersions.py** – implementation of the getOCPVersions API which will quickly return a list of APIs leveraging a Redis Cache.
- **loadOCPVersions.py** – Kubernetes cronjob that will load current versions of OCP supported by IBM Cloud into a Redis database.
- **getOCPToken.py** – implementation of the real-time getOCPToken API.
- **enableNodeSSH.py** – implementation of the 'long' running enableNodeSSH API

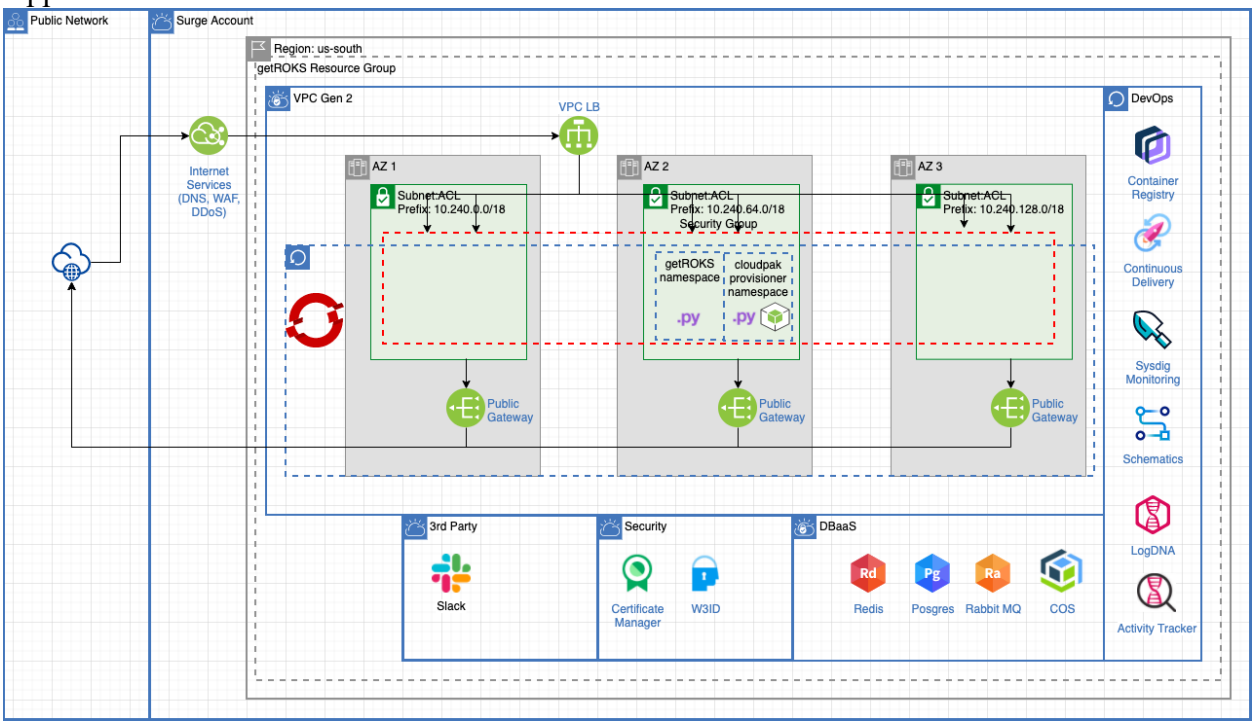
Example Enterprise Application

To help guide the discussion, I'm going to step through a real-world cloud-native application that I helped develop that is being used across IBM. The reason for both a tutorial and a real-world application is because many times a tutorial is too simplistic. When you are developing a real-world enterprise ready application there are many issues you face that a tutorial doesn't cover.

I'll point out some of the additional complexities to consider while you are developing an enterprise level application. The tutorial is meant for those how may have not developed on IBM Cloud to get hands on experience. I will share lessons learned through discussion of the enterprise application.

The cloud native application that I will go through the development lifecycle of actually has two applications one called getROKS and other called CloudPak Provisioner. The getROKS application was developed to provide an automated web application that allows IBMers to request and provision an IBM Cloud Managed Openshift cluster to use for a proof of concept with our clients. CloudPak Provisioner is an API based application that provides easy to use APIs to simplify and automate common tasks that are performed with RedHat OpenShift and IBM Cloud Paks running on IBM Cloud. Both applications leverage many different services from the IBM Catalog that each section of this series will go through. Each topic will include the architecture decisions on why certain services were used, implementation details on how to use the IBM Cloud Services and lessons learned. You will notice that the tutorial is based on the CloudPak Provisioner application.

Application Architecture:



CloudPak Provisioner

Cloud Pak Provisioner is a collection of APIs that make it easy to perform common tasks for managing and configuring ROKS clusters that is available for anyone to use as it is exposed on the Internet. Many of the APIs are focused on IBM CloudPaks but can be extended to any deployment of Managed OpenShift on IBM Cloud. You are more than welcome to try out any of the CloudPak Provisioner APIs.

If you would like to try out the CloudPak Provisioner APIs, I have created a Postman Collection that you can import.
Postman Collection: https://www.getpostman.com/collections/bc63c30083ec77b2c3c5

You can also view more detailed documentation on using the APIs here:
https://github.com/kmcolli/CloudPakProvisioner

API Reference

There are seven categories of APIs:

- IBM Cloud Object Storage
- IBM CloudPak for Data
- VPC Infrastructure
- General Utility
- Classic Infrastructure
- Portworx
- OpenShift

IBM Cloud Object Storage.

Retrieve a list of COS Buckets

Retrieves a list of buckets including metadata for the bucket and the size of the bucket. Search criteria includes being able to search by name and metadata.

Delete COS Bucket and all Objects

Single API call that deletes all objects in a bucket and the bucket itself.

Add Metadata to COS Bucket

Adds metadata to a bucket by creating a file called README file in the bucket that contains key value pairs. Metadata given to the API call will overwrite the existing metadata.

Retrieve a List of COS Metadata for a Bucket

Retrieves a list of metadata for a bucket by reading a file name README containing key value pairs.

Copy a COS Bucket to Another Bucket

Single API that directly copies one bucket to another in the same COS instance.

Create a COS PVC on a ROKS Cluster

This api will do the following:

- Install the ibm-object-storage helm chart on a ROKS Cluster
- Create a COS bucket if not already created
- Create a secret in ROKS to acess the COS Bucket
- Create a pvc attached to the ROKS cluster

CloudPak for Data.

Add Certificates to Cloud Pak for Data

- Creates a dns entry in cloud internet services mapping cp4d-<cluster_name>.ibmcloudroks.net to the cp4d console route that references the cluster ingress subdomain.
- updates CP4D nginx pods with ibmcloudroks.net certificates
- updates the cp4d route to point to the secured URL

Update Certificates for Cloud Pak for Data

- updates the CP4D nginx pods with ibmcloudroks.net certificates

Run CP4D Pre-installation Script

- runs the cp4d pre-installation script before you can install cp4d with IBM Cloud Schematics
- optimizes CP4D by setting linux worker node parameters no root squash and setting the kernel semaphore.

VPC.

Attach raw block devices to ROKS Worker Nodes on a VPC

- creates raw block devices on VPC Gen 2 in each zone the worker nodes are in
- attaches the created block devices to ROKS worker nodes
- works in both MZR and SZR

Delete raw block devices to ROKS Worker Nodes on a VPC

- unattaches block devices created with the *attachBlockVPC* api.
- deletes block devices

Get list of block devices attached to ROKS Worker Nodes on a VPC

- retrieves a list of block devices attached to worker nodes on a VPC.

Classic Infrastructure.

Remove unused block volumes on classic infrastructure

- removes all block volumes not attached to a Kubernetes cluster
- *warning - this is destructive and will immediate delete all block volumes that are not being used by Kubernetes. Only run this if you are sure you are not using block volumes outside of Kuberenetes

---

Portworx.

Provision portworx using the IBM Cloud Catalog

- Creates a portworx instance for a given cluster
- Portworx instance is created in the same resource group as the cluster
- Assumes raw block storage devices are already created.

Create Block Storage on Classic

- Installs the IBM Block Attacher Helm chart into the given cluster
- For each worker node in the cluster the following is performed:
    - orders a block device using the maximum IOPS for the given size
    - authorizes the worker node to use the block device
    - attaches the block device to the worker node
    - creates a PV for the block device

Utility

Check if ROKS Cluster is ready

Checks a ROKS cluster to see if the ingress subdomain has been set for the cluster and the cluster details populate the resource group name. After the cluster is provisioned, it can take up to 30 minutes for the cluster to recognize the resource group name it was create in.

OpenShift.

Get ROKS Versions

Using a local Redis cache, this api quickly returns a list of versions available in ROKS.

Get ROKS Token

Retrieves an ROKS token and login command by providing your api key and cluster name.