Module 2 - Lab 1 Assessing the App and Environment for Containers

Lab Worksheet

In this exercise, you will evaluate BioGuide and the environment in which BioGuide currently operates to determine your path to containerizing the workload and deploying it to your existing Kubernetes environment. Using the lab guide, complete this worksheet.

Understanding the Goal

1. Why is the application being migrated?
2. Can the behavior of the application be modified? If not, can the cloud environment be modified to accommodate the application?

Gather Information about the Application

Determine Network Interactions

The network is by far the most common interface in modern applications. Consider both the outbound connections that the application initiates and the inbound connections that are initiated by other applications. These interfaces must be preserved during migration so that applications can function properly.

1. What must the application connect to in order to function?

|  |  |  |  |
| --- | --- | --- | --- |
| Description | URL or IP Address | Port | Inside or Outside of the Cluster |
| Example: Azure Blob Storage | myblob.example.com | 1234 | Outside |
|  |  |  |  |
|  |  |  |  |

1. What services or users need access to the application?

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Protocol | Port | Inside or Outside of the Cluster |
| Example: SOLR Search container | https | 443 | Inside |
|  |  |  |  |
|  |  |  |  |

Determine Filesystem Interactions

Modern applications use filesystems for storage of configuration, static data, and dynamic data. Historically, filesystems were also used as a mechanism for communication between applications and components. Although this practice has largely been replaced by network-based interaction, it is still present in some domains and in legacy applications. Filesystem interactions should be noted, however, because they are very important to the migration.

1. What files and directories are read by the service?

For each file or directory, determine whether the content is static, configuration, or dynamic (i.e., modified at runtime).

|  |  |
| --- | --- |
| Directory/File | Static/Configuration/Dynamic |
| Example: /etc/app/config.yaml | Configuration |
|  |  |
|  |  |

Use the Collected Data to Plan the Migration

If proceeding with the migration is still deemed valuable, the actual migration involves two steps:

1. Containerization of all processes that make up the application.
2. Selecting the Kubernetes objects that will make up the components of the application in the new environment.

Step 1 will be covered later in this workshop. To proceed with step 2, here are some questions to help determine which Kubernetes objects are needed for migration of the application.

1. What controller should be used to manage the Pods?

Kubernetes has several built-in controllers for managing Pods, and each exhibits slightly different behavior.

**Replicaset** -- A Replicaset ensures that a specified number of Pods is running at any point in time. If you define that there should be five Pods running, the Replicaset will make sure that this happens. If there are any excess Pods, they get deleted.

**Deployment** -- A Deployment controller is used to run a Pod at a desired number of replicas. These Pods have no unique identities. The Deployment can specify the configuration over a standalone Pod and a Replicaset, such as what deployment strategy to use. For example, if you are upgrading an application from v1 to v2, you might consider one of the following approaches:

* Upgrade with zero downtime
* Upgrade sequentially one after the other
* Pause and resume upgrade process
* Rollback upgrade to the previous stable release

**DaemonSet** -- Like Deployment, except the Pod will be started once on every node.

**StatefulSet** Deployment controllers are suitable for managing stateless applications. Statefulsets, on the other hand, are useful when running workloads that require persistent storage. They keep unique identities for each Pod they manage and use the same identity when Pods need to be rescheduled.

**Job** -- A Kubernetes Job is a controller that supervises Pods for carrying out certain tasks. They are primarily used for batch processing. As soon as you submit a Job manifest file to the API server, the Pod will kick in and execute a task. When the task is completed, it will shut down by itself.

1. How will persistent storage be achieved?

If persistent storage is required for the application, then first consider the use of cloud-native storage options in your environment (e.g., Blob Storage in Azure). However, doing so will require changes to the application to support the use of these systems instead of the filesystem.

If it is not feasible to change the application, then a PersistentVolume is the way to go. If possible in your environment, a LocalPersistentVolume is a good last-resort option if performance of network-attached storage is not sufficient.

3. How should configuration be injected?

Configuration should be injected into the Pods via a ConfigMap. However, data such as passwords and keys should be injected via a Secret instead using a secrets repository such as Azure Key Vault.

4. How should static files, such as certificates, be managed?

Static data should be compiled into the Docker image that makes up the Pods. Alternatively, it could be treated the same as persistent storage and loaded once before deployment.

5. How should communication between components via filesystems be handled?

A PersistentVolume can usually be created with the necessary access permissions and mounted into the Pods that need to communicate. However, the PersistentVolume implementation that is used for a volume impacts whether that volume can be used by multiple readers or writers simultaneously.

The best way to solve this problem is to avoid it by restructuring the application to use network-based communication mechanisms. If restructuring the application is not possible (or not desirable) due to business constraints, then the processes that are communicating must be placed in the same Pod and communicate via an emptyDir volume.

6. How will log data be read from Pods?

Kubernetes expects all Pods to log everything worth logging to STDOUT and STDERR. If the application does not support the ability to modify logging options, then a sidecar container (e.g., fluentd or logstash) can be used to read logs from a shared volume and transfer them into a log aggregation system.

Microsoft Azure has Azure Monitor, a set of logging and monitoring services, which can be enabled for each Azure service. Two of those services, Application Insights and Container Insights, perform application and AKS cluster monitoring respectively.

7. Does the Kubernetes environment support network policy?

If the Kubernetes environment supports network policy, then the host and port information gathered should be used to create a Network Policy object.