**Kubernetes Migration Step by Step**

The high-level steps for migration plan include

1. Making the App Kubernetes Ready
2. Establish the CI/CD process
3. Prepare Kubernetes Cluster.

**Making App Kubernetes Ready**

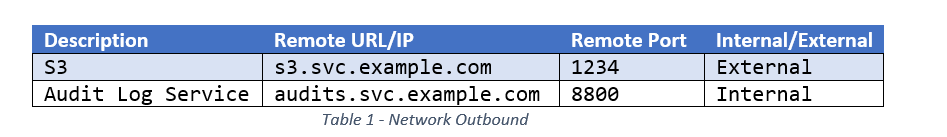
1. Review the current architecture with the various components involved and consider the components to be migrated
2. Consider the communications options between the components whether message queues, service buses or API calls.
3. Rethinking some components might be helpful but keeping to the current architecture and focusing on the migration helps not complicating the journey.
4. Containerize the application using ready to use Docker images for many frameworks and application runtimes. So many examples and documents on how to Dockerize your component for each modern programming language.
5. Map to K8’s objects, choose from the list of options provided by Kubernetes to host the components including deployments, services, endpoints, jobs and other components. The best practice is to come up with a K8s architecture choosing the objects for your components.
6. Next step is to create the Kubernetes objects for your components. Writing YAML files for Kubernetes deployment, services, pods, jobs and other components. Creating a separate repository for these configurations utilizing GitOps increasing the reliability and security of our K8’s resources.
7. Most cases leave the database as is and connect to the database from the Kubernetes application.

**Gather information about the system**

**Determine Network Interactions**

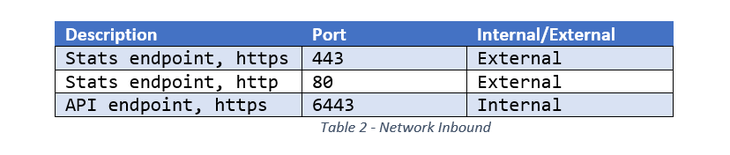
1. What must the service connect to inorder to function?

Make sure to record the URL or IP, the port and the location of the resource within or outside the cluster.



1. What services or users need access to this service?

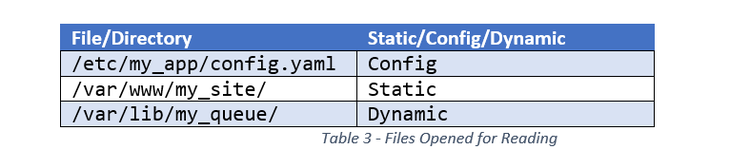
Make sure to record the port and location of the remote user/service inside the cluster and outside the cluster or both.



**Determine File Interactions**

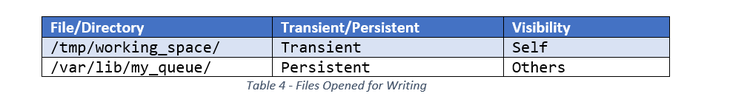
**What files and directories are read by the service?**

For each file or directory determine whether the content is static, configuration or dynamic



**What files are modified by the service ?**

Make sure to record whether the modifications should survive service restarts i.e to be persistent or transient. Also record whether the changes must be visible to other applications or executables.



**Use the data to make a suitable decision**

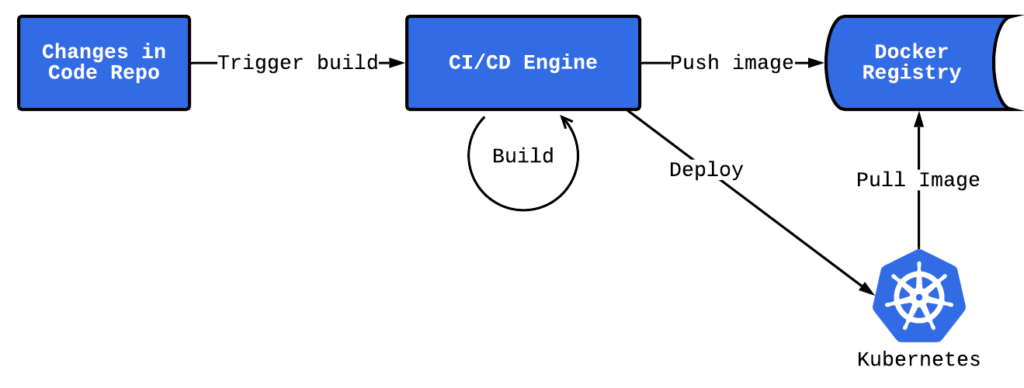
After gathering the above information, it gives a deep understanding of the application which can be used to make a decision whether to migrate the application or not. Once it deemed that the migration is worth the effort the migration process involves

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

**Establish the CI/CD process:**

This stage is the most important stage, is to configure a delivery pipeline for your changes to Kubernetes. The desired CI/CD process is defined in the diagram below:

1. Establishing a CI/CD process starts with new changes in the source code repository.
2. Then, the CI/CD engine starts and executes the build process.
3. The outcome of the build is a ready to deploy Docker image with the application files. The image is pushed to the Docker repository.
4. The CI/CD engine fires the deploy necessary by updating the image version for the corresponding K8s deployment.
5. Kubernetes updates pods and pulls a new application image from the Docker repository.



**Configure Build runners** Choose build runners which support the creation of Docker images and be able to build your app and pack it into the image. It would be better to have a Docker runner where each build is executed inside the Docker.

**How to containerize the applications?**

**Build the container image**

Docker builds images by reading instructions from a Docker file. A docker file is a simple text file that contains instructions that can be executed on the command line. Using command line, the docker build we can start a build that executes all of the command line

**Select an Azure Container Registry**

Create an Azure container registry to build and store the images.

**Configure CI/CD builds**

Here you should setup the CI/CD process. Need to include the configuration files inside each repo that describe the build process. Start configuration as a code process for the first application part and reuse the scripts for other parts. Have a separate source code repository to store, build and deploy templates for commonly used applications and components.

**Create Kubernetes deployment scripts**

The last step in to trigger the Kubernetes Admin API, then initiate the deploy process. Usually Python or shell scripts are used this.

For example for Python there are some Kube API client libraries that you can use for Kubernetes access and its resource manipulation. The goal of these scripts is to update the corresponding K8’s deployment object with a new image version and Kubernetes will do the rest. First start testing these scripts in the local K8s setup or in a dev cluster.

**Deploy the containerized app on AKS**

Once the image is built it is required to deploy the application as a container on AKS (Azure Kubernetes Service).

1. Select the Azure Kubernetes Services Cluster

Specify the AKS cluster that the application should be deployed to. The selected AKS cluster must have a Linux node pool.

The cluster has to be configured to allow pulling of the images from the Azure Container Registry that was selected to store the images.

Run the following command in Azure CLI to attach the AKS cluster to the ACR.

az aks update -n <cluster-name> -g <cluster-resource-group> --attach-acr <acr-name>

Kubernetes Cluster configuration is a complex process that involves several milestones. Best practice advice is to start with a simple version and evolve it later. Don’t try and do too much from the start, choose only the components you require to migrate effectively.

Please consider the following for your cluster:

1. A friendly control panel to use. You need an admin panel to manage all Kubernetes parts. It depends on how you create your cluster; some tools already provide you with a nice UI for K8s management
2. Configures security and policies. Establish cluster security and organize user access control.
3. Network traffic and DNS. Networking is an essential part of the cluster. Starting with some basic network policies then setup ingress traffic and including DNS configuration.
4. Cluster monitoring. Gather metrics and configure corresponding alerts on them. The standard combination here is Prometheus + Grafana, or some ready made products like New Relic, Dynatrace and Datadog and others.
5. Available volume storage. Most of the applications have some files to store. In this case you should provide them with some persistent, resilient storage.
6. Specify secret store: If you had opted to parameterize application configurations, then specify the secret store to be used for the application. You can choose Azure Key Vault or Kubernetes secrets for managing your application secrets. Sensitive data like connection strings and passwords etc can be stored more effectively.

Monitoring Workspace: If you’d selected enabled monitoring with Application Insights, then specify the application insights resource that we want to use.

1. Centralized log storage: To know what is going on in our services the logging is the most crucial part of the application. Can establish our own solution or use some existing products.
2. Kubernetes namespaces: Need to think about how to organize your cluster resources. Will work in the default namespaces or somehow create multiple namespaces with their own resource quotas.

1. Specify the Azure file share. If you had added more folders and selected the persistent volume option, then specify the Azure file share that should be used by Azure migrate: App containerization tool during the deployment process. The tool will create new directories in this Azure file share to copy over the application folders that are configured for Persistent Volume storage. Once the application deployment is complete, the tool will clean up the Azure file share by deleting the directories it had created.
2. Application deployment configuration: Once you have completed the steps above, you’ll need to specify the deployment configuration for the application. You can provide the following customizations:

Prefix String

Replica sets

Load Balancer Type

Application configuration

Storage

1. Deploy the application: Once the deployment configuration for the deployment is saved, the tool will generate the Kubernetes deployment YAML for the application.

The desired result here is to have all Kubernetes configuration as a source code which is compliant with the IaC (Infrastructure as a code) approach. Often, we can have a separate repository to store scripts for all Kubernetes objects. Start with plain Kubernetes scripts and then combine them into Helm Charts. You can easily recreate your cluster in any environment.

