May 2021

Hybrid Azure DevOps Infrastructure for Validation

Rachana Palacharla
Schlumberger

Hybrid Azure Devops Infrastructure for Validation

Rachana Palacharla, Schlumberger

In a cloud empowered software development ecosystem, continuous testing is crucial to speed up the idea to production pipeline. In a distributed software development, early testing avoids failures caused by dependencies. At the same time testing can be enhanced using the automation. The foundation for manual or automated testing is the underlying infrastructure. The boom in containerization has given us a lot of light weight options to be able to run automated tests. But containers are not always the solution to providing a test environment that mimics the production environment. This document provides a solution to manage virtual machines by using the infrastructure as code method. All the steps in this solution have been automated as well.

The motivation behind creating this system for automating the creation of a clean validation server came from a few issues that were slowing down the productivity of our team.

1. The fact that the Azure testing servers, available to all the teams globally, were extremely unstable. At times different teams pushed breaking changes that impacted us or vice versa. The servers were rarely cleaned up so the results were inconsistent too.
2. Due to the location of various teams in different time zones, there was no support for failures happening on those shared Azure testing servers during our time zone.
3. We tried to get ourselves an Azure server, but this presents cost issues.
4. We also set foot in the container world and used Docker but soon ran into issues with testing the frontend on certain browser versions.

Prepping the Image:

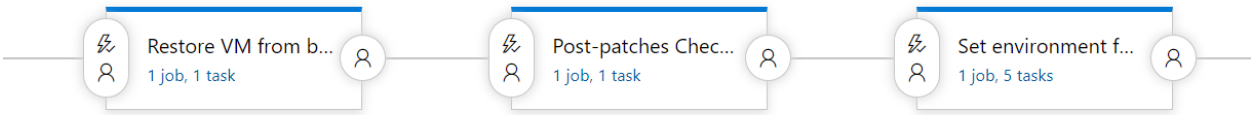
We owned a powerful physical machine that was available as a tool. I consolidated all the VMs on it and created a new virtual machine using the IT standard Windows server. My approach from the beginning was to script any of the steps done manually with it. Some steps had issues, but most steps could be executed using PowerShell. I list a few below. I saved all the necessary software on a shared drive on the host for easy access to the scripts.

1. Changing registry settings.
2. Adding necessary users and admins.
3. Installing browsers and other software like .NET and VSTools, Service Fabric cluster, Git, Node, Consul etc.
4. Enable IIS settings, RabbitMQ, Redis and Mongo DB.
5. Install patches.
6. Download and install the Azure Devops agent.

Installing SQL server had issues at times so that was manual. But once these pre-requisites were installed, I created a snapshot of the server and marked it as the base-image. Then I started to work with scripts to install software with all the components to start testing. The server had more than 200 service fabric services running on it. This did not perform well even on a powerful host. I painfully moved out other major VMs off of this host so this server could have majority of the resources on the host. At times the pre-installed software needs to be upgraded or removed. That has to be done manually on top of this base-image.

Image checkpoint management:

The idea here is to automate the resetting of the server image to a default point that is ready to deploy the latest software without any remnants of previous installations. I created an Azure Devops release pipeline that deployed to one or more deployment servers that were prepped using the technique mentioned above. The pipeline had 2 main stages (shown in the picture below) that would bring the server to a good default state.



The first stage "Restore VM from backup" uses the Virtual machine checkpoint management commands to restore to a previous base image that has a clean Image without the product software installed on it. To be specific, this will create a backup of the current state of current VM with a specified naming pattern and then restore to the previously saved latest base image snapshot. The name of the image was automatically created with the same naming pattern.

The second stage is kicked off only after 8 hours. In my case, I reset the VM on Friday night and run the second stage “Post-patches Checkpoint” on Sunday noon. This is done so that all the IT patches that this old image is missing are installed by IT checks. In my case the image would be 1 week old, and it will prompt a restart before snapshotting this new clean base image.

Third stage is to load up the environment variables needed for our software product. After which this release pipeline triggers another separate pipeline which is used to deploy the software on a clean image. The details of this pipeline are explained in the next topic. But this step guarantees a new version of the software will be available first thing in the morning for our team to use.

I also scripted deleting old images periodically to save space on the host as the images can easily fill up the space.

Automation of clean Image:

Once the server is ready and patched the other customizations can be performed through scripts. I used another release pipeline to run these scripts as they could be run multiple times for different configurations of the software by testers or developers. Customizations for us included the following.

- Loading the static data into the databases.
- Downloading the Feature toggles based on the configuration chosen.
- Uploading the custom flags to Consul
- Deploying the software with all the service fabric services.
- Re-starting the Service fabric cluster and other services.

There have been many occasions the teams depended on this server to validate the new features when the shared Azure servers were not available. There were some issues with networking or the

changes in the software that had prompted this server to not have a good deployment of the software. But that has changed since the complexity of the software has been simplified.