# Multicloud Deployment

### Problem

Global firms are frequently required to serve a global audience. As a result, their apps must be available across several areas, each with its own set of compliance standards.

Some parts of the Application, or the entire Application, may need to be deployed to an environment that runs in a specific area.

For example, we may run an application in both an Asian and European data centre, but some Asian countries, such as China, require it to be run in a China data centre.

Another scenario is deploying a distributed application that makes use of native cloud services from many cloud vendors. This implies we'll need to be able to deploy the other services to a variety of cloud vendors, including AWS, Google Cloud Platform, and Azure.

In a worldwide organisation, there is also a requirement to deploy to several clouds. The programme, for example, choose Azure, while another selects the Internal cloud, GCP, or both.

### Solution

Deploy a modular pipeline that can be used everywhere, is simple to set up, and allows DevOps teams to deploy to various internal and external clouds.

#### Options

##### Serverless Pipeline

Terraform will be used to create a serverless pipeline. We can start with one cloud (Azure) and subsequently expand to other clouds.

Azure Pipelines is a service that allows you to build pipelines on the Azure Cloud Platform. It offers continuous integration (CI) and continuous delivery (CD), which means it tests and builds your code on a regular basis and deploys it to any target via a pipeline.

An example of a Terraform Script to establish an Azure Pipeline is shown below.

|  |
| --- |
| terraform { |
|  | required\_providers { |
|  | azuredevops = { |
|  | source = "microsoft/azuredevops" |
|  | version = ">=0.1.0" |
|  | } |
|  | } |
|  | } |
|  |  |
|  | provider "azuredevops" { |
|  | org\_service\_url = var.devops\_org\_service\_url |
|  | personal\_access\_token = var.devops\_personal\_access\_token |
|  | } |
|  |  |
|  | resource "azuredevops\_project" "janethterraformproject" { |
|  | name = var.project |
|  | description = var.project\_description |
|  | visibility = var.visibility |
|  | version\_control = var.version\_control |
|  | work\_item\_template = var.work\_item\_template |
|  | features = { |
|  | "boards" = "enabled" |
|  | "repositories" = "enabled" |
|  | "pipelines" = "enabled" |
|  | "testplans" = "enabled" |
|  | "artifacts" = "enabled" |
|  | } |
|  | } |
|  |  |
|  | resource "azuredevops\_serviceendpoint\_github" "github\_serviceendpoint" { |
|  | project\_id = azuredevops\_project.janethterraformproject.id |
|  | service\_endpoint\_name = var.project\_github\_service\_endpoint\_name |
|  |  |
|  | auth\_personal { |
|  | personal\_access\_token = var.github\_personal\_access\_token |
|  | } |
|  | } |
|  |  |
|  | resource "azuredevops\_build\_definition" "tf" { |
|  | project\_id = azuredevops\_project.janethterraformproject.id |
|  | name = var.build\_definition\_name |
|  | ci\_trigger { |
|  | use\_yaml = true |
|  | } |
|  | repository { |
|  | repo\_type = "GitHub" |
|  | repo\_id = join("/", [var.github\_org\_name, var.github\_repo\_name]) |
|  | branch\_name = var.build\_branch |
|  | yml\_path = var.project\_build\_yml\_dir\_path |
|  | service\_connection\_id = azuredevops\_serviceendpoint\_github.github\_serviceendpoint.id |
|  | } |
|  | } |

The Azure DevOps Services enable for many cloud deployments in tandem.

The following YAML snippet demonstrates how to deploy to clusters in parallel from several clouds. Deployments are made to resources corresponding to namespaces in AKS, GKE, EKS, and OpenShift clusters in this example. Under the 'contoso' environment, these four namespaces are associated with Kubernetes resources.

trigger:

- main

jobs:

- deployment:

displayName: Deploy to AKS

pool:

vmImage: ubuntu-latest

environment: contoso.aksnamespace

strategy:

runOnce:

deploy:

steps:

- checkout: self

- task: KubernetesManifest@0

displayName: Deploy to Kubernetes cluster

inputs:

action: deploy

kubernetesServiceConnection: serviceConnection #replace with your service connection

namespace: aksnamespace

manifests: manifests/\*

- deployment:

displayName: Deploy to GKE

pool:

vmImage: ubuntu-latest

environment: contoso.gkenamespace

strategy:

runOnce:

deploy:

steps:

- checkout: self

- task: KubernetesManifest@0

displayName: Deploy to Kubernetes cluster

inputs:

action: deploy

kubernetesServiceConnection: serviceConnection #replace with your service connection

namespace: gkenamespace

manifests: manifests/\*

- deployment:

displayName: Deploy to EKS

pool:

vmImage: ubuntu-latest

environment: contoso.eksnamespace

strategy:

runOnce:

deploy:

steps:

- checkout: self

- task: KubernetesManifest@0

displayName: Deploy to Kubernetes cluster

inputs:

action: deploy

kubernetesServiceConnection: serviceConnection #replace with your service connection

namespace: eksnamespace

manifests: manifests/\*

- deployment:

displayName: Deploy to OpenShift

pool:

vmImage: ubuntu-latest

environment: contoso.openshiftnamespace

strategy:

runOnce:

deploy:

steps:

- checkout: self

- task: KubernetesManifest@0

displayName: Deploy to Kubernetes cluster

inputs:

action: deploy

kubernetesServiceConnection: serviceConnection #replace with your service connection

namespace: openshiftnamespace

manifests: manifests/\*

- deployment:

displayName: Deploy to DigitalOcean

pool:

vmImage: ubuntu-latest

environment: contoso.digitaloceannamespace

strategy:

runOnce:

deploy:

steps:

- checkout: self

- task: KubernetesManifest@0

displayName: Deploy to Kubernetes cluster

inputs:

action: deploy

kubernetesServiceConnection: serviceConnection #replace with your service connection

namespace: digitaloceannamespace

manifests: manifests/\*

Pros

👍 ***Categorized Built in Tasks****—*The tasks are categorized based on the nature of operation. ex: Build tasks, Utility tasks, Deploy tasks etc. This makes easy for the user to add the desired/specific tasks to their pipeline.

👍 ***Group Tasks* —**It allows you to encapsulate a sequence of tasks, already defined in a pipeline, into a single reusable task ,just like any other task.

👍 ***Configuring CI/CD Pipeline as Code****—*Using YAML we can achieve this. [**Reference to understand the hierarchy of YAML file**](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/integrate-jenkins-pipelines-cicd?view=azure-devops&tabs=yaml)-

👍 ***Request and Add Tasks to your Pipelines***— It has a lot of build-in tasks, yet you can download extensions/tasks from the Azure DevOps marketplace.

👍 ***Microsoft Hosted Agents****—*Azure Pipelines offers cloud hosted build agents for Linux, Windows, and macOS builds. You can have a look as to what software are installed on the agent using [**the following reference link**](https://github.com/microsoft/azure-pipelines-image-generation/tree/master/images/macos).

👍 ***Any language, any platform, any cloud****—*Build, test, and deploy Node.js, Python, Java, PHP, Ruby, C/C++ , .Net, Android, and iOS apps. Run in parallel on Linux, macOS and Windows. Deploy to Azure, AWS, GCP or on-premises

👍 ***Azure Pipelines****—*Provide unlimited build minutes to all open source projects and up to [**10 concurrent jobs across Windows, Linux and macOS**](https://azure.microsoft.com/en-ca/blog/announcing-azure-pipelines-with-unlimited-ci-cd-minutes-for-open-source/).

👍 ***Azure Pipeline****—*Analytics are provided at the end of each run with parameters like rate and duration of run.

## Cons

👎 **Integration with non-Microsoft is difficult** — Azure DevOps should provide easier integration with other product lines to improve acceptability.

👎 **Azure Pipeline** **—**Workflow is straightforward (can’t set if-else or switch-case constructions). This makes it more difficult to develop complex workflows.

👎 The **deprecated tasks/extensions**— They are not removed from the marketplace.

##### Jenkins

A Jenkins pipeline is a set of plugins that help you set up and use continuous delivery pipelines in Jenkins.

Jenkins remains one of the most effective Swiss Army knives for the job. I believe that anything is feasible with Jenkins, however it may take a little more effort and 3 dozen plugins at times. As systems integrators, we're frequently confronted with previously unidentified requirements that arise at the last minute. Adopting instruments that provide "escape hatches" gives us "peace of mind" in the knowledge that we can fix any difficulty.

Parts of it, such as the GUI Configurations, feel a little old, but this is offset by Configuration as Code and GitOps. I wish some tasks, such as constructing and operating Docker containers inside Kubernetes pipelines, were simpler. Let's be honest. Jenkins is no longer the cool kid on the block, and there are a plethora of excellent tools available. But the truth is that few will last as long as Jenkins has in the Open Source and Enterprise worlds.

###### Pros

👍 Jenkins is **open source**, **easy to install** and **free**.

👍 **Platform Independent**— Available for all platforms and different operating systems, whether Mac OS X, Windows or Linux.

👍 **Rich Plugin**— Jenkins comes with a wide range of plugins.

👍 Blue Ocean + Material Theme for Jenkins makes it look like any other modern CI/CD system

👍 Rich ecosystem of Plugins enables the ultimate level customization, much more than any SaaS

👍 Easily extract metrics from your build system into a platform like Prometheus. Centralize your monitoring of things running inside of CI/CD infrastructure. This is very difficult (or not even possible) to do with many SaaS offerings.

👍 Arguably the Jenkinsfile Declarative Pipelines DSL is very readable, in fact, it looks a lot like HCL (HashiCorp Configuration Language). To some, this will be a “Con” – especially if YAML is a requirement.

👍 Jenkins “Configuration as Code” plugin supports nearly everything you'd need to run Jenkins itself in a “GitOps” compliant manner. And if it doesn’t there is always the [configuration-as-code-groovy plugin](https://github.com/jenkinsci/configuration-as-code-groovy-plugin) which allows you to run arbitrary Groovy scripts for the bits you need (credit: Steve Boardwell).

👍 Jenkins supports what seems like an unlimited number of credential backends. This is a big drawback with most SaaS-based CI/CD platforms. With the Jenkins credential backends, it's possible to “plug and play” things like “AWS SSM Parameter Store”, “AWS Secrets Manager” or HashiCorp Vault.

###### Cons

👎 **Outdated UI**— Its interface seems a bit outdated and not user friendly as it doesn’t follow modern design principles.

👎 **Scripted Pipelines** — Must be programmed in **Groovy**.

👎 Though it’s rich in Plugin, a lot of **plugins are not straightforward or unstable** — Even for a basic tasks, plugins needs to be installed.

👎 There’s no**YAML interface** for Jenkins Pipelines.

👎 Jenkins doesn’t provide any**analytics**(there are plugins but they are not enough) at the end of each run.

👎 Needs better documentation.

👎 [Jenkins has many plugins that seem no longer maintained](https://jenkins.io/doc/developer/plugin-governance/adopt-a-plugin/). It's important to make sure whatever plugins you chose are still receiving regular updates (as in something pushed within the last ~12 months).

👎 Not all plugins are compatible with Declarative Pipelines. IMO using Declarative Pipelines is the current gold standard for Jenkins. Raw imperative groovy pipelines are notoriously complicated and unmanageable.

👎 No less than a few dozen plugins are required to “modernize” Jenkins. The more plugins, the greater the chance there will be problems during upgrades. This can be somewhat mitigated by moving towards using command-line driven tools run inside containers as opposed to installing some of the more exotic plugins (credit: Steve Boardwell).

👎 There's no (maintained) YAML interface for Jenkins Pipelines (e.g. Jenkinsfile.yaml). Most modern [CI](https://cloudposse.com/glossary/continuous-integration/)/[CD](https://cloudposse.com/glossary/continuous-delivery/) platforms today have adopted YAML for pipeline configuration. In fact, Jenkins X has also moved to YAML.

👎 It's not cloud-based.

👎 Have to maintain infrastructure yourself.

👎 Jenkins doesn’t provide any analytics (there are plugins but they are not enough) on the end-to-end deployment cycle. This again goes back to the lack of overall tracking that contributes to the lack of analytics as well.

👎 Sometimes a bit slow

👎 Sometimes installing the wrong plugins ends up with Jenkins in a non-startable state.

👎 When there is a huge number of builds, loading the Jenkins UI takes minutes. Sometimes times out as well.

👎 There were cases where jobs were in a hung state and could not be aborted as well. Jenkins restart was the only solution.

##### Jenkins X

Jenkins X will automatically create beautiful pipelines for your projects that implement completely CI and CD, so you don't have to know the inner workings of Jenkins Pipeline.

Pros

* Environment Promotion via GitOps - Each team gets a set of Environments. Jenkins X then automates the management of the Environments and the Promotion of new versions of Applications between Environments via GitOps
* Pull Request Preview Environments - Jenkins X automatically spins up Preview Environments for your Pull Requests so you can get fast feedback before changes are merged to master
* Feedback on Issues and Pull Requests - Jenkins X automatically comments on your Commits, Issues and Pull Requests with feedback as code is ready to be previewed, is promoted to environments or if Pull Requests are generated automatically to upgrade versions
* Kubernetes integration
* Scripted Pipelines
* GitOps
* MLOps
* Jenkinsx can be run also as serverless

Cons

* Complexity

Spinnaker

**Spinnaker** is detailed as "Multi-cloud continuous delivery platform for releasing software changes with high velocity and confidence". Created at Netflix, it has been battle-tested in production by hundreds of teams over millions of deployments. It combines a powerful and flexible pipeline management system with integrations to the major cloud providers.

Conclusion

In my opinion we should try the serverless pipeline and also Jenkins X.

If we want to use Jenkins I would extend an existing internal initiative that uses JSON files to drive the pipeline.